

Реальные проблемы и решения

2-е издание

Java™ и XML



O'REILLY®

Бретт Мак-Лахлин

Java & XML

Second Edition

Brett McLaughlin

O'REILLY®

Java и XML

Второе издание

Бретт Мак-Лахлин



*Санкт-Петербург
2002*

Бретт Мак-Лахлин

Java и XML

Перевод Т. Морозовой

Главный редактор
Зав. редакцией
Научный редактор
Редактор
Корректура
Верстка

А. Галунов
Н. Макарова
М. Зислис
В. Овчинников
С. Беляева
А. Дорошенко

Мак-Лахлин Б.

Java и XML. – Пер. с англ. – СПб: Символ-Плюс, 2002. – 544 с., ил.
ISBN 5-93286-018-9

Java и XML. Эти две технологии уже давно привлекают внимание разработчиков. И не зря. Они идеально подходят для создания веб-ориентированных корпоративных приложений, обеспечивают независимость от платформы, расширяемость, возможность повторного использования кода, а также поддержку стандарта Unicode. Их соединение позволяет создавать веб-сайты с динамически обновляемыми страницами, разрабатывать корпоративные приложения, снижающие затраты на совместное использование информации, и находить простые и эффективные решения проблемы переносимости данных. Автор описывает применение всего арсенала инструментов и средств XML и Java. Здесь и DTD и пространства имен, XML Schema и XPath, XSL и различные API (SAX, DOM, JDOM). Рассматривается связывание данных, разработка приложений при помощи XML-RPC и SOAP, использование систем веб-публикации (например, Apache Cocoon). Не оставлены вниманием создание веб-служб с применением SOAP, UDDI и WSDL, каналы RSS, динамические данные и XSP.

Второе издание «Java и XML» дополнено главами о расширенных возможностях SAX и DOM, а также о SOAP и связывании данных. Эта книга станет незаменимым спутником для тех, кто пишет программы на Java и собирается применять XML (или планирует заниматься этим), участвует в движении peer-to-peer, разрабатывает программное обеспечение для электронной коммерции либо использует службу сообщений или веб-службы.

ISBN 5-93286-018-9

ISBN 0-596-00197-5 (англ)

© Издательство Символ-Плюс, 2002

Authorized translation of the English edition © 2001 O'Reilly & Associates Inc. This translation is published and sold by permission of O'Reilly & Associates Inc., the owner of all rights to publish and sell the same.

Все права на данное издание защищены Законодательством РФ, включая право на полное или частичное воспроизведение в любой форме. Все товарные знаки или зарегистрированные товарные знаки, упоминаемые в настоящем издании, являются собственностью соответствующих фирм.

Издательство «Символ-Плюс». 193148, Санкт-Петербург, ул. Пинегина, 4,
тел. (812) 324-5353, edit@symbol.ru. Лицензия ЛП N 000054 от 25.12.98.

Налоговая льгота – общероссийский классификатор продукции
ОК 005-93, том 2; 953000 – книги и брошюры.

Подписано в печать 27.05.2002. Формат 70x100¹/₁₆. Печать офсетная.

Объем 34 печ. л. Тираж 2000 экз. Заказ N

Отпечатано с диапозитивов в Академической типографии «Наука» РАН
199034, Санкт-Петербург, 9 линия, 12.

Оглавление

Предисловие	9
Структура этой книги	9
Для кого предназначена эта книга?	12
Программное обеспечение и версии	13
Типографские соглашения	14
Комментарии и вопросы	14
Благодарности	15
1. Введение	17
XML имеет значение	17
Что важно?	20
Основы	23
Что дальше?	26
2. Основы технологии	27
Основы	28
Ограничения	40
Преобразования	48
И еще...	58
Что дальше?	58
3. SAX	59
Подготовительные операции	60
SAX-совместимые анализаторы	62
Обработчики содержимого	69
Обработчики ошибок	87
Советы разработчикам	93
Что дальше?	97
4. Расширенный SAX	98
Свойства и возможности	99
И опять обработчики	107
Фильтры и объекты Writer	113
И снова обработчики	120

Советы разработчикам	125
Что дальше?	128
5. Объектная модель документа	129
Объектная модель документа	129
Сериализация	135
Изменяемость	148
Советы разработчикам	149
Что дальше?	151
6. Расширенный DOM	152
Изменения	153
Пространства имен	164
Модули DOM Level 2	168
DOM Level 3	183
Советы разработчикам	187
Что дальше?	188
7. JDOM	189
Основы	189
Класс PropsToXml	195
Класс XMLProperties	206
Является ли JDOM стандартом?	217
Советы разработчикам	219
Что дальше?	221
8. Расширенный JDOM	222
Полезная информация по внутренней организации JDOM	222
JDOM и фабрики	228
Классы Wrapper и Decorator	233
Советы разработчикам	246
Что дальше?	248
9. JAXP	249
API или абстракция	249
JAXP 1.0	251
JAXP 1.1	260
Советы разработчикам	270
Что дальше?	272
10. Системы веб-публикации	273
Выбор системы публикации	275
Установка	278
Применение системы публикации	283

XSP	300
Сосоон 2.0 и выше	316
Что дальше?	319
11. XML-RPC	320
RPC и RMI: за и против	321
Простейшее приложение	324
Переложение нагрузки на сервер	337
Реальные ситуации	353
Что дальше?	356
12. SOAP	357
Начинаем	357
Установка	361
«Пачкаемся»	366
Идем дальше	375
Что дальше?	384
13. Веб-службы	385
Веб-службы	385
UDDI	387
WSDL	389
Собираем все вместе	392
Что дальше?	411
14. Объединение содержимого	412
Библиотека Foobar	413
Компания mytechbooks.com	423
Рассылка или запрос	433
Что дальше?	443
15. Связывание данных	444
Основные принципы	446
Castor	452
Zeus	460
JAXB	469
Что дальше?	477
16. Взгляд в будущее	478
XLink	478
XPointer	480
Отображения для схем XML	485
И прочее...	486
Что дальше?	487

A. Справочник по API	488
SAX 2.0	488
DOM Level 2	501
JAXP 1.1	510
Пакет javax.xml.transform.stream	516
JDOM 1.0 (Beta 7)	516
Пакет org.jdom	516
B. Возможности и свойства SAX 2.0	528
Основные возможности	528
Базовые свойства	530
Алфавитный указатель	532

Предисловие

Когда я чуть больше года назад писал предисловие к первому изданию «Java и XML», я даже и не представлял себе, с чем связываюсь. Я отпуская шуточки о том, что XML скоро появится на шапках и футболках; сейчас, когда я пишу эти строки, на мне футболка с вышитой на ней надписью «XML», а еще у меня есть кепка с надписью XML (на самом деле у меня их даже две!). Так что XML, без сомнения, оправдал все ожидания. И это хорошо.

Но это означает, что каждый день ведутся новые разработки, и ландшафт XML расширяется с такой скоростью, которую я не мог себе представить даже в самых смелых предположениях. И хотя для XML это здорово, при обращении к первому изданию это ввергает в уныние: почему все настолько устарело? Я говорил о SAX 2.0 и DOM Level 2 как о чем-то новом. Теперь это промышленные стандарты. Я только знакомил читателей с JDOM, а теперь он входит в JSR (Sun's Java™ Specification Request, запрос на разработку или изменение спецификации Java-технологии). Я даже не рассматривал SOAP, UDDI, WSDL и связывание данных XML. В этом издании им отведено три главы! Все изменилось, и это не преувеличение.

Даже тем, кто только подозревает, что в ближайшие несколько месяцев, возможно, будет работать с XML, эта книга, весьма вероятно, будет полезна. А если где-то на вашем рабочем столе есть первое издание этой книги, то просмотрите и новое издание; думаю, вы обнаружите, что оно все же актуально. Я избавился от чрезмерных описаний основных концепций, собрал материал об основах XML в одной главе и переписал практически все примеры. Кроме того, я добавил много новых примеров и глав. Другими словами, я попытался создать подробную, многогранную техническую книгу. Начинаящим на ее изучение потребуется немного больше времени, поскольку помощи стало меньше, зато знаний будет гораздо больше.

Структура этой книги

Эта книга очень четко структурирована: в ее первой половине – главах с 1 по 9 – основное внимание уделяется изучению основ XML и ключевых интерфейсов прикладного программирования Java для обработки XML. Каждому из трех интерфейсов для работы с XML (SAX, DOM

и JDOM) автор посвящает по главе, в которой рассматривает базовые понятия, и еще по главе, описывающей более сложные концепции. Глава 10 – переходная, здесь начинается подъем по «стеку» XML. В ней рассматривается JAXP, представляющий собой уровень абстракции над SAX и DOM. Остальная часть книги – главы с 11 по 15 – посвящена специфическим вопросам, которые постоянно поднимаются в конференциях и руководствах, с которыми я связан, и предназначена для желающих научиться применять XML в своих приложениях. Среди этих тем новые главы по SOAP, связыванию данных и новый взгляд на приложения business-to-business. Наконец, есть еще два приложения, завершающие книгу. Итак, содержимое книги таково:

Глава 1 «Введение»

Посмотрим, о чем, собственно, шум, изучим начала XML и уделим время обсуждению причин, по которым XML настолько важен для настоящего и будущего корпоративных разработок.

Глава 2 «Основы технологии»

Это ускоренный курс основ XML, от XML 1.0 до DTD и от XML Schema до XSLT и пространств имен. Для тех, кто читал первое издание, это сумма всех глав, посвященных работе с XML, и кое-какой дополнительный материал.

Глава 3 «SAX»

В этой главе вводится и рассматривается простой API для XML (SAX), первый Java API для обработки XML. Подробно рассматривается жизненный цикл процесса анализа, а затем демонстрируются события, которые SAX может перехватывать и которые могут использоваться разработчиками.

Глава 4 «Расширенный SAX»

Здесь мы еще глубже познакомимся с SAX, рассматривая реже используемые, но тем не менее очень мощные элементы API. Узнаем, как применять фильтры XML для объединения в цепи обратных вызовов, как применять классы XMLWriter и DataWriter для вывода XML при помощи SAX. Кроме того, мы рассмотрим некоторые из менее распространенных обработчиков SAX, такие как LexicalHandler и DeclHandler.

Глава 5 «DOM»

В этой главе читатель движется дальше по ландшафту XML к следующему API Java и XML – DOM (Document Object Model, объектная модель документа). Вы изучите основы DOM, выясните, что находится в текущей спецификации (DOM Level 2), и узнаете, как читать и создавать деревья DOM.

Глава 6 «Расширенный DOM»

Продолжая изучать DOM, вы узнаете о различных модулях DOM, например Traversal, Range, Events, CSS и HTML. Также мы посмотр-

рим, что собой предоставляет новая версия DOM Level 3 и как применять эти новые возможности.

Глава 7 «JDOM»

Эта глава содержит введение в JDOM и описывает его отличия от DOM и SAX, не оставляя без внимания их сходства. Рассматривается чтение и создание XML при помощи этого API.

Глава 8 «Расширенный JDOM»

Углубляя знакомство с JDOM, мы рассмотрим практические приложения API – интеграцию с JAXP и то, как JDOM обеспечивает использование фабрики с вашими собственными подклассами JDOM. Кроме того, вы увидите, как XPath действует в паре с JDOM.

Глава 9 «JAXP»

В настоящее время JAXP – это развившийся API, поддерживающий анализ и преобразования, и он заслуживает отдельной главы. Тут мы рассмотрим обе версии – 1.0 и 1.1, и вы узнаете, как использовать этот API во всей полноте.

Глава 10 «Системы веб-публикации»

В этой главе рассказано, чем является система веб-публикации, почему она имеет к вам отношение и как выбрать хорошую систему. Затем мы остановимся на системе Apache Cocoon, более внимательно рассмотрим множество ее свойств и то, как она может применяться для создания динамического наполнения веб-страниц.

Глава 11 «XML-RPC»

Здесь рассматриваются удаленные вызовы процедур (Remote Procedure Calls, RPC), их уместность в распределенном окружении по сравнению с RMI и то, как XML делает RPC жизнеспособным решением некоторых проблем. Затем мы перейдем к применению Java-библиотек XML-RPC и созданию клиентов и серверов XML-RPC.

Глава 12 «SOAP»

В этой главе мы остановимся на использовании данных конфигурирования в формате XML и увидим, почему этот формат столь важен для кроссплатформенных приложений, особенно по отношению к распределенным системам и веб-службам.

Глава 13 «Веб-службы»

Продолжая говорить о SOAP и веб-службах, в этой главе мы познакомимся с двумя важными технологиями – UDDI и WSDL.

Глава 14 «Объединение содержимого»

Не покидая руслу межкорпоративных (business-to-business) приложений, эта глава представляет еще один способ взаимодействия приложений – при помощи соглашений о содержимом. Вы узнаете о Rich Site Summary, создании информационных каналов и даже немного о Perl.

Глава 15 «Связывание данных»

И снова вверх по стеку XML. В этой главе говорится об одном из высокоуровневых интерфейсов для Java и XML, а именно интерфейсе связывания данных XML. Вы узнаете, что такое связывание данных, как оно позволяет существенно упростить работу с XML, а также осознаете существующие возможности. Я рассмотрю три системы: Castor, Zeus и недавнюю версию JAXB (Java Architecture for XML Data Binding, архитектура Java для связывания данных XML) от Sun.

Глава 16 «Взгляд в будущее»

В этой главе я указываю на несколько интересных технологий, появившихся на горизонте, и привожу немного дополнительной информации по каждой из них. Некоторые из моих догадок могут быть совершенно неверными, другие же могут оказаться действительно чем-то интересным.

Приложение А «Справочник по интерфейсам прикладного программирования»

В приложении подробно описаны все классы, интерфейсы и методы, доступные в интерфейсах SAX, DOM, JAXP и JDOM.

Приложение В «Возможности и свойства SAX 2.0»

В этом приложении подробно рассмотрены возможности и свойства, доступные в реализациях синтаксических анализаторов, поддерживающих SAX 2.0.

Для кого предназначена эта книга?

Эта книга основана на предпосылке, что XML очень быстро становится (и в некоторой степени уже стал) важным аспектом программирования на Java. Материал книги учит вас применять XML и Java, и нигде, кроме как в первой главе, не задается вопрос «*следует ли применять XML*». Разработчикам на Java несомненно следует использовать XML. Вот почему эта книга адресована программистам на Java, а также тем, кто собирается программировать на Java, имеет в подчинении программистов на Java либо связан с проектом, в котором используется Java. Если вы хотите совершенствоваться как разработчик, создавать более понятный код или хотите, чтобы проект был завершен вовремя и не вышел за рамки бюджета, если вам необходим доступ к данным устаревших систем, если вам нужно распределение системных компонентов либо если вы просто хотите знать, почему XML уделяется столько внимания, то эта книга для вас.

Я старался как можно меньше представлять себе конкретного читателя; я не верю, что следует усложнять введение в XML настолько, чтобы невозможно было войти в курс дела. Тем не менее, я также пола-

гаю, что если вы потратили деньги на эту книгу, то вам нужны не просто основы. Поэтому я предполагал лишь, что вы знакомы с языком Java и понимаете основные концепции серверного программирования (скажем, из области Java-сервлетов и Enterprise JavaBeans™). Если вы никогда прежде не писали на Java или только начинаете изучать язык, то, возможно, вам стоит прочесть книгу Пэта Нимайера (Pat Niemeyer) и Джонатана Кнудсена (Jonathan Knudsen) «Learning Java», O'Reilly (Изучаем Java), прежде чем приступать к этой книге. Я не предполагаю, что вы знаете что-либо об XML, поэтому начинаю с простого. Но я полагаю, что вы хотите хорошо поработать и быстро научиться; поэтому базовые понятия мы рассмотрим быстро, а основной объем книги будет посвящен работе с более сложными вещами. Материал не повторяется, если нет явной необходимости, так что кому-то, возможно, понадобится перечитать предшествующие разделы или полистать страницы в обоих направлениях, поскольку рассмотренные ранее понятия используются в последующих главах. У тех, кто хочет изучить XML, немного знаком с Java и готов набирать код примеров в своем любимом текстовом редакторе, не должно возникнуть особых проблем при чтении этой книги.

Программное обеспечение и версии

В этой книге рассмотрен стандарт XML версии 1.0 и различные словари XML по состоянию на июль 2001 года. Поскольку многие затронутые спецификации XML еще не приняты в окончательном варианте, могут существовать незначительные несоответствия между печатным изданием данной книги и текущей версией обсуждаемой спецификации.

Весь приводимый код на Java основан на платформе Java 1.2. Если вы пока еще не пользуетесь Java 1.2, постарайтесь это сделать – одни только классы коллекций стоят этого. Синтаксический анализатор Apache Xerces, процессор Apache Xalan, библиотека Apache SOAP и библиотеки Apache FOP применялись в виде самых свежих доступных стабильных версий на июнь 2000 года, а система веб-публикации Apache Cocoon – в версии 1.8.2. Версия использованных Java-библиотек XML-RPC – 1.0 beta 4. Все обсуждаемое программное обеспечение распространяется свободно, и его можно получить в сети по адресам <http://java.sun.com>, <http://xml.apache.org> и <http://www.xml-rpc.com>.

Исходные тексты примеров из этой книги полностью содержатся на ее страницах. Как исходные тексты, так и скомпилированные версии всех примеров (включая объемную документацию Javadoc, не обязательно включенную в текст книги) доступны в сети по адресам <http://www.oreilly.com/catalog/javaxml2> и <http://www.newInstance.com>. Все примеры, которые могут выполняться в качестве сервлетов либо могут быть преобразованы для выполнения в таком качестве, можно просмотреть и использовать, посетив страницу <http://www.newInstance.com>.

Типографские соглашения

В этой книге приняты следующие типографские соглашения:

Курсивом выделяются:

- Имена путей Unix, имена файлов и имена программ
- Адреса Интернета, включая доменные имена и URL
- Определения новых терминов

Полужирным шрифтом выделены:

- Имена элементов GUI: заголовки окон, кнопки, пункты меню и т. д.

Моноширинным шрифтом выделены:

- Командные строки и параметры, которые необходимо ввести дословно
- Имена и ключевые слова в программах Java, включая названия методов, имена переменных и имена классов
- Имена элементов и тегов XML, имена атрибутов и других конструкций XML, приводимых в том виде, в каком они присутствуют в XML-документе

Комментарии и вопросы

Пожалуйста, направляйте замечания и вопросы относительно этой книги издательству:

O'Reilly & Associates, Inc.
101 Morris Street
Sebastopol, CA 95472
(800) 998-9938 (в США или Канаде)
(707) 829-0515 (международный/местный)
(707) 829-0104 (факс)

Также можно отправить сообщение по электронной почте. Чтобы попасть в наш список рассылки или заказать каталог, воспользуйтесь адресом

info@oreilly.com

Чтобы задать технический вопрос или прокомментировать содержание книги, напишите по адресу

bookquestions@oreilly.com

Мы поддерживаем веб-сайт, посвященный этой книге, на котором будут приведены примеры, замечания о найденных ошибках и планы переизданий. Эта страница расположена по адресу

<http://www.oreilly.com/catalog/javaxml2/>

Более подробную информацию об этой и других книгах можно найти на веб-сайте издательства O'Reilly:

<http://www.oreilly.com>

Благодарности

Вот и снова я пишу благодарности. Сейчас вспомнить каждого ничуть не проще, чем в первый раз. Мой редактор Майк Лукидес (Mike Loukides) даже по ночам беспокоится о том, как сделать все правильно, и именно таким должен быть хороший редактор! Кайл Харт (Kyle Hart) – суперженщина из отдела маркетинга – ведет всю работу и напоминает мне о том, что в конце туннеля виден свет. Тим О'Рейли (Tim O'Reilly) и Фрэнк Уиллисон (Frank Willison) терпеливы, но настойчивы, и именно такими должны быть хорошие руководители. А Боб Экштейн (Bob Eckstein) и Марк Лой (Marc Loy) помогали мне разрешать досадные проблемы с графическим интерфейсом Swing. (Кроме того, Боб просто забавный парень. Посмотрим правде в глаза.) O'Reilly – настолько хорошее издательство, насколько это возможно, во всех отношениях. Я горжусь этим сотрудничеством.

Также хочу поблагодарить бесподобную команду моих рецензентов. Много раз они просматривали главу меньше чем за 24 часа, при этом выдавая добросовестный отчет по техническим деталям. Именно эти люди внесли весомый вклад в поддержание технического уровня книги. Роберт Сиз (Robert Sese), Филип Нельсон (Philip Nelson) и Виктор Брилон (Victor Brilon) – все они просто изумительны. Конечно же, я всегда готов благодарить своего «соучастника» Джейсона Хантера (Jason Hunter) за то, что он предан JDOM и другим техническим вопросам (старина, устрой себе выходной на одну ночь!). Наконец, компания, в которой я работаю – Lutris Technologies, – это, пожалуй, лучшее место, в котором можно пожелать работать. Они позволили мне много часов работать над книгой и не жаловались ни разу. В особенности я хочу отметить Янси Линд (Yancy Lind), Пола Моргана (Paul Morgan), Дэвида Юнга (David Young) и Кейт Бигелов (Keith Bigelow) – они просто мастера своего дела. Спасибо!

Снова спасибо моим родителям – Ларри (Larry) и Джуди Мак-Лахлин (Judy McLaughlin). Я люблю вас за то, что вы вырастили в меру честного и целеустремленного сына (вы, конечно, понимаете, что эти характеристики также относятся и к невыносимому ребенку!). Сара Джейн (Sarah Jane) – моя тетя, и мои бабушка и дедушка Дин (Dean) и Глэдис Мак-Лахлин (Gladys McLaughlin) никогда не допускали даже и мысли о том, что я о них не думаю только из-за того, что вижу их редко. Дедушка, ты даже не догадываешься, насколько я благодарен тебе за то, что ты собираешься посмотреть на второе издание. Я люблю вас всех.

Спасибо моим вторым родителям (родителям жены), Гари (Gary) и Ширли Грейтхаус (Shirley Greathouse) – вы просто лучше всех. Когда-нибудь я соберусь и объясню, что вы значите для меня, но это может потребовать отдельной книги. Я люблю вас обоих за ваш юмор и мудрость. Спасибо Куину (Quinn) и Джони (Joni) за ту легкость, которую они привносят в наши воскресные обеды. Спасибо Лонни и Лауре, не могу дождаться, когда увижу Беби Джей (Baby J). Спасибо Биллу и Терри за то, что они были очень мудрыми друзьями, а также спасибо Биллу за то, что он такой пастор, какого еще нужно поискать.

Источник смеха в моей жизни – это несколько веселых людей, и я просто не могу не упомянуть их тут: Кендра (Kendra), Британи (Brittany), Лизетт (Lisette), Джаней (Janay), Роки (Rocky), Дастин (Dustin), Тони (Tony), Стефани (Stephanie), Робби (Robbie), Эрин (Erin), Анджела (Angela), Майк (Mike), Мэтт (Matt), Карлос (Carlos) и Джон (John). Я увижусь с вами в воскресенье, и, пожалуйста, давайте не пойдем больше в Mazzio's. Также спасибо моим собакам: Сету, Чарли, Джейку, Мозесу, Молли и Дэйзи. Тот не жил по-настоящему, кого не будил по утрам холодный язык бассет-хаунда.

Наконец, спасибо двум людям, которые значат для меня гораздо больше, чем остальные – моему дедушке Роберту Эрлу Бердону, с которым я вновь встречаюсь однажды. Я думаю о тебе каждый день, и мои дети тоже вскоре о тебе услышат. Но больше всего я хочу поблагодарить мою жену Ли. Словами этого не выразить. Однажды все песни и слезы, связанные с тобой, выплеснутся наружу, и ты наконец-то поймешь, как много для меня значишь.

Наконец, спасибо Господу, который дал мне достичь столь многого.

- *Выбор системы публикации*
- *Установка*
- *Применение системы публикации*
- *XSP*
- *Socoop 2.0 и выше*
- *Что дальше?*

10

Системы веб-публикации

С этой главы начинается обзор специальных тем, посвященных Java и XML. Мы уже рассмотрели основы применения XML в Java, уделили внимание использованию API SAX, DOM, JDOM и JAXP для работы с XML, а также вопросам использования и создания XML-документов. Теперь, когда читатели понимают, как использовать XML в собственном коде, можно перейти к обсуждению конкретных приложений. В следующих шести главах (главы 10–15) представлены важнейшие приложения XML и, в частности, их реализации в мире технологий Java. Хотя уже существуют тысячи важнейших приложений XML, темы, рассматриваемые в этих главах, постоянно находятся в центре внимания, и в них заложен значительный потенциал для изменения традиционных процессов разработки приложений.

Первая актуальная тема, которую мы рассмотрим, связана с применением XML, которое вызвало наибольший резонанс среди разработчиков на XML и Java. Речь идет о системе веб-публикации. Хотя я постоянно говорю, что визуализацию содержимого документа, возможно, переоценивают по сравнению со значимостью переносимости данных, которую обеспечивает XML, применение XML для формирования представления данных все же является очень важным. Потребность в визуализации возрастает, когда речь заходит о веб-ориентированных приложениях.

Практически каждое серьезное приложение, которое я видел, либо полностью основано на веб-технологиях либо, как минимум, имеет веб-интерфейс. В то же время, пользователи требуют больше функциональности, а отделы маркетинга – больше гибкости в пользовательском интерфейсе. В результате растет количество веб-оформителей.

Чем больше все меняется, тем больше остается неизменным

Те, кто читал первое издание книги, обнаружат, что большая часть материала по Сосооп в этой главе осталась прежней. Хотя я и обещал, что к этому времени выйдет Сосооп 2, и собирался написать целую главу, посвященную ему, все развивалось не так быстро, как я того ожидал. Стефано Маццокки (Stefano Mazzocchi) – главная движущая сила Сосооп – решил, наконец, закончить учебу (так держать, Стефано!), и в результате разработка Сосооп 2 замедлилась. Основная разработка по-прежнему сосредоточена на Сосооп 1.x, так что ее и следует использовать. Раздел, посвященный Сосооп 2, обновлен и отражает нововведения. Следите за новыми книгами от O'Reilly, посвященными вопросам, связанным с системой Сосооп.

Эта новая профессия отличается от веб-мастера только тем, что в профессиональные обязанности художника не входит знание Perl, ASP, JavaScript и других языков создания сценариев. Весь рабочий день веб-оформителя посвящен созданию, изменению и разработке документов HTML и WML.¹ Быстрые изменения в деловой активности и рыночной стратегии могут требовать полной переработки приложения или сайта с частотой до раза в неделю, и зачастую веб-оформитель вынужден тратить целые дни на переделку сотен HTML-страниц. Хотя каскадные таблицы стилей (Cascading Style Sheets, CSS) помогают в этом деле, поддержание согласованности всех этих страниц требует огромного количества времени. Но даже если считать эту далекую от совершенства ситуацию приемлемой, ни один разработчик захочет тратить свою жизнь на внесение бесконечных изменений в разметку веб-страниц.

С приходом серверных технологий Java эта проблема лишь усугубилась. Оказалось, что разработчикам сервлетов приходится тратить многие часы на переделку операторов `out.println()`, которые отвечают за вывод конструкций HTML, и они обычно исподлобья смотрят в сторону отдела маркетинга, когда выясняется, что изменения в оформлении сайта требуют внесения изменений в код приложения. Есть основания считать, что следствием этой ситуации явилась целая спецификация Java Server Pages (JSP). Однако технология JSP не решает проблемы, она лишь перекладывает головную боль на автора напол-

¹ Языки разметки HTML и WML используются совместно с сопутствующими технологиями, такими как Shockwave Flash. Грамотное применение подобных технологий далеко не тривиально, так что я никоим образом не преуменьшаю роль этих людей.

нения страниц, которому постоянно приходится проявлять осторожность, чтобы по ошибке не внести изменения во внедренный код Java. К тому же, технология JSP не обеспечивает четкого разделения содержимого и представления, т. е. не решает поставленной задачи. Требовалось получить средство создания содержимого, свободного от посторонних данных, а также средство согласованного оформления содержимого – либо в заданные моменты времени (*статическое создание наполнения*), либо динамически во время выполнения запроса (*динамическое создание наполнения*).

Те, кто участвовал в разработке веб-приложений, наверняка кивнули, узнав знакомую проблему, и есть надежда, что ваши мысли обращены к технологиям XSL и XSLT. Проблема вот в чем: должна существовать система для управления созданием наполнения документов, в особенности, если речь идет о динамическом создании наполнения. Бессмысленно иметь сайт, состоящий из сотен XML-документов, если отсутствует механизм, обеспечивающий преобразование, когда запрашивается тот или иной документ. Добавьте к этому потребность сервлетов и других серверных компонентов в выводе XML-данных, которые должны быть единообразно оформлены, и вы получите минимальный набор требований, предъявляемых к системе веб-публикации. В этой главе мы рассмотрим такую систему, посмотрим, как она позволяет экономить многие часы возни с HTML-разметкой. Эта система позволит веб-оформителям стать знатоками XML и XSL, сделав пользовательский интерфейс приложений максимально гибким и легко изменяемым.

Системы веб-публикации стремятся решить эти сложные задачи. Подобно тому как веб-сервер несет ответственность за передачу файла в ответ на запрос для URL этого файла, система веб-публикации ответственна за реагирование на схожие запросы. Однако вместо того чтобы возвращать сам файл, система веб-публикации обычно возвращает *публикуемую* версию файла. Публикуемый файл отличается от исходного тем, что мог подвергнуться преобразованиям с помощью XSLT, мог быть изменен приложением либо преобразован в иной формат, такой как PDF. Отправитель запроса не видит «необработанные» данные, которые могут стоять за публикуемым результатом, но и не должен явным образом запрашивать выполнение преобразования для публикации. Как правило, базовый URI (скажем, <http://yourHost.com/publish>) указывает на то, что запросы должна обрабатывать система публикации, установленная поверх веб-сервера. Как вы, возможно, догадываетесь, эта концепция существенно проще, чем фактическая реализация такой системы публикации, и поиск системы, подходящей для конкретных задач, является отнюдь не простой задачей.

Выбор системы публикации

Возможно, читатель надеется обнаружить список, содержащий информацию о сотнях возможных решений. Как вы могли видеть, язык

Java и различные API для XML предоставляют простые средства для создания приложений XML. Помимо этого, простым решением обработки запросов и создания ответов в веб-пространстве могут оказаться сервлеты Java. Реальность же такова, что перечень систем веб-публикации невелик, а если оставить лишь качественные и стабильные, он станет еще меньше. Один из лучших источников информации о доступных в настоящее время продуктах – это список, публикуемый на сайте XML Software, <http://xmlsoftware.com/publishing/>. Этот список обновляется достаточно часто, и поэтому нет смысла приводить его в этой книге. Тем не менее, следует упомянуть некоторые важные критерии выбора системы, оптимальной для конкретных задач.

Устойчивость

Не удивляйтесь, если вам (по-прежнему!) трудно найти продукт, номер версии которого выше 2.x. В действительности, наверное, надо как следует поискать, чтобы найти систему хотя бы второго поколения. И хотя более высокий номер версии отнюдь не гарантирует стабильной работы, обычно он отражает время и усилия, затраченные на производство продукта, а также количество переработок кода, которые претерпела система. Системы публикации XML являются настолько новым явлением, что рынок заполнен продуктами версий 1.0 и 1.1, которые просто недостаточно устойчивы для практического применения.

Зачастую можно убедиться в устойчивости продукта, изучив другие разработки того же производителя. Обычно производитель выпускает целый набор средств, и если некоторые из продуктов не обеспечивают поддержки SAX 2.0 и DOM Level 2 либо все имеют версии 1.0 и 1.1, было бы разумно воздержаться от применения такой системы до тех пор, пока она не станет более совершенной и не будет соответствовать новейшим стандартам XML. Также старайтесь держаться подальше от технологий, ориентированных на какую-либо конкретную платформу. Если система веб-публикации жестко привязана к операционной системе (например, Windows или даже к определенному варианту Unix), это значит, что вы имеете дело с решением, которое построено не только на Java. Помните, что системы веб-публикации предназначены для клиентов, работающих на любой платформе. Зачем же пользоваться продуктом, который сам не может работать на любой платформе?

Интеграция с другими XML-продуктами и API

Убедившись, что ваша система публикации достаточно устойчива для решения ваших задач, удостоверьтесь, что она поддерживает широкий диапазон XML-анализаторов и процессоров. Если система жестко привязана к определенному анализатору или процессору, то вы буде-

те ограничены какой-то одной реализацией технологии. А это плохо. Хотя системы публикации обычно хорошо интегрированы с анализатором вполне определенного производителя, выясните, являются ли анализаторы взаимозаменяемыми. Проверьте, можно ли будет использовать ваш любимый (или оставшийся от предыдущих проектов) процессор.

Поддержка SAX и DOM является обязательной, а многие системы веб-публикации в настоящее время также поддерживают JDOM и JAXP. Даже если у вас есть любимый API, чем больше возможностей вы имеете, тем лучше! Попробуйте также найти систему, разработчики которой пристально следят за спецификациями схем XML, XLink, XPath и других словарей XML. По этому признаку вы поймете, можно ли ожидать появление поддержки этих технологий в новых версиях системы (а это важный признак ее долговечности). Не стесняйтесь спрашивать, как скоро следует ожидать включения новых спецификаций в продукт, и настаивайте на четком ответе.

Факт применения

Последний и, вероятно, самый важный вопрос, на который надо ответить при выборе системы веб-публикации, – применяется ли она в реальных приложениях. Если вам не могут указать хотя бы несколько приложений или ссылок на сайты, которые используют данную систему публикации, не удивляйтесь, если их вообще не существует. Производители (или разработчики, если говорить о мире приложений с открытым кодом) должны с радостью и чувством гордости сообщать, где можно увидеть их систему в действии. Колебания в этой ситуации – признак того, что при освоении этого продукта вы можете стать большим первопроходцем, чем вам того хотелось бы. Например, Apache Cocoon предоставляет такой список по адресу <http://xml.apache.org/cocoon/livesites.html>.

Принятие решения

Оценив продукты по перечисленным критериям, скорее всего, можно получить ясный результат. Очень немногие системы публикации могут дать положительные ответы на все вопросы, поставленные здесь, не говоря уже о требованиях, предъявляемых вашим приложением. На самом деле, к июлю 2001 года существовало меньше десяти систем публикации, поддерживающих самые последние версии SAX (версия 2.0), DOM (Level 2) и JAXP (версия 1.1), используемых при этом хотя бы на одном сайте и претерпевших по меньшей мере три существенные переработки кода за время своего существования. Их список здесь не приводится по той причине, что, откровенно говоря, в течение полугода они могут исчезнуть или измениться радикальным образом.

Мир систем веб-публикаций развивается столь динамично, что я скорее введу вас в заблуждение, если рекомендую четыре или пять возможных вариантов, полагая, что они будут доступны через несколько месяцев.

Но есть одна система публикации, которая пользуется неизменным успехом в сообществе разработчиков Java и XML. Если говорить о разработках с открытым кодом, разработчики на Java часто обращаются именно к этой системе. Проект Apache Socoop, начатый Стефано Маццокки, привел к созданию надежной системы с самого начала. Система Socoop создавалась уже в то время, когда большинство из нас лишь пытались разобраться в том, что же такое XML, и сегодня разрабатывается второе поколение этой системы, основанной полностью на Java. Эта система является также частью проекта Apache XML и по умолчанию поддерживает Apache Xerces и Apache Xalan. Она позволяет использовать любой анализатор XML, соответствующий стандарту, и основана на чрезвычайно популярной архитектуре сервлетов Java. Кроме того, существует несколько сайтов, построенных на Apache Socoop (в версии 1.x), которые расширяют границы традиционной разработки веб-приложений и при этом показывают исключительно хорошую производительность. По этой причине, и, как всегда, следуя принципам открытого исходного кода, я остановил свой выбор в этой главе на Apache Socoop.

В предыдущих главах выбор анализатора и процессора XML был достаточно произвольным. Иначе говоря, с очень незначительными изменениями в коде примеры должны были работать при использовании разработок других производителей. Тем не менее, системы веб-публикации еще не стандартизованы, и каждая из них реализует самые разные возможности и следует разным соглашениям. Приводимые в этой главе примеры работают только для Apache Socoop и не являются переносимыми. Тем не менее, популярность концепций и архитектурных решений, использованных в Socoop, заслуживает отдельной главы. Если ваш выбор пал на другую систему публикации, следует, по крайней мере, взглянуть на приводимые здесь примеры, потому что концепции применимы для всех реализаций, хотя код таковым и не является.

Установка

В других главах инструкции по установке сводились к адресу сайта в Интернете, где можно получить дистрибутив, и добавлению *jar-файла* в переменную CLASSPATH. Установка системы публикации вроде Socoop – задача, вовсе не столь простая, поэтому рассмотрим процедуру в подробностях. Кроме того, на сайте Socoop приведены инструкции по установке, касающиеся различных сред исполнения

сервлетов; их можно найти по адресу <http://xml.apache.org/cocoon1/install.html>.¹

Исходный код или скомпилированные версии

Прежде всего, следует определить, в каком виде нужна система Coooon: в исходных кодах или в скомпилированном виде. Решение можно свести к следующему вопросу: вам нужны самые последние возможности или максимально надежная версия? Если вы активный разработчик и хотите разобраться с Coooon, установите инструмент CVS, а затем извлеките последнюю версию исходного кода Coooon из CVS-репозитория на xml.apache.org. Чтобы не расписывать подробно собственно процесс, который будет интересен далеко не всем читателям, сошлюсь на книгу Грегора Парди (Gregor Purdy) «*CVS Pocket Reference*» (O'Reilly). Этой книги, а также инструкций, приведенных на странице <http://xml.apache.org/cvs.html>, будет вполне достаточно.

Если стоит цель оценить работу системы Coooon или использовать ее в реальных приложениях, следует загрузить скомпилированную версию с сайта <http://xml.apache.org/cocoon/dist/cocoon1>.² Как уже было сказано, версия 1.8.2 доступна для Windows (*Cocoon-1.8.2.zip*) и для Linux/Unix (*Cocoon-1.8.2.tar.gz*). Получив архив, распакуйте его во временный каталог, в котором можно работать. Отметим одну важную деталь – при этом создается каталог *lib/*. Этот каталог содержит все библиотеки, необходимые для запуска Coooon в среде исполнения сервлетов.

Примечание

Если каталог *lib/* отсутствует либо не содержит нескольких *jar*-файлов, то у вас, возможно, более старая версия Coooon. Только новые версии (начиная с 1.8) содержат эти библиотеки (которые, кстати, существенно облегчают жизнь!).

Настройка среды исполнения сервлетов

После сборки системы Coooon следует настроить среду исполнения сервлетов для совместной работы с Coooon и указать, какие запросы должны передаваться Coooon. Рассмотрим настройку работы системы Coooon со средой сервлетов Jakarta Tomcat.³ Поскольку она является

¹ Основной сайт Coooon расположен по адресу <http://xml.apache.org/cocoon>, а последней версией считается 2.0. С корневой страницы основного сайта есть ссылка на сайт Coooon 1.x, который, соответственно, находится в каталоге *cocoon1*. – *Примеч. науч. ред.*

² В *.../dist/* доступна лишь ветка 2.0, ветка 1.x хранится в подкаталоге *cocoon1*. – *Примеч. науч. ред.*

³ Страница проекта Tomcat доступна по адресу <http://jakarta.apache.org/tomcat/>. – *Примеч. науч. ред.*

эталонной реализацией для Java Servlet API версии 2.2, проблем с повторением этих шагов для произвольной среды исполнения сервлетов быть не должно.

Прежде всего, следует скопировать все библиотеки, необходимые для работы Cоsoon, в каталог библиотек Tomcat, *TOMCAT_HOME/lib*, где *TOMCAT_HOME* – это каталог, в который установлен Tomcat. На моей системе Windows это каталог *c:\java\jakarta-tomcat*, а в Linux – */usr/local/jakarta-tomcat*. Не весь каталог *lib/* Cоsoon подлежит копированию; для выполнения Cоsoon требуются следующие *jar*-файлы:

- *bsfengines.jar* (система сценариев Bean)
- *bsf.jar* (система сценариев Bean)
- *fop_0_15_0.jar* (FOP)
- *sax-bugfix.jar* (исправления SAX в части обработки ошибок)
- *turbine-pool.jar* (Turbine)
- *w3c.jar* (W3C)
- *xalan_1_2_D02.jar* (Xalan)
- *xerces_1_2.jar* (Xerces)

Кроме того, скопируйте в тот же каталог (*TOMCAT_HOME/lib*) файл *bin/cocoon.jar*. Теперь у вас есть все библиотеки, необходимые для запуска Cоsoon.

Последние версии Tomcat (я пользуюсь версией 3.2.1) автоматически загружают все библиотеки из каталога Tomcat *lib/*, то есть вам не придется мучиться с путями к классам. Если среда не поддерживает автоматическую загрузку, добавьте все *jar*-файлы в пути к классам среды сервлетов.

После добавления необходимых библиотек остается указать среде исполнения сервлетов *контекст* для работы Cоsoon, то есть объяснить, где искать файлы, запрашиваемые через Cоsoon. Делается это путем изменения файла *server.xml* из каталога Tomcat *conf/*. Добавьте следующую инструкцию в конец этого файла, в элемент *ContextManager*:

```
<Server>
  <!-- Прочие элементы Server -->

  <ContextManager>
    <!-- Прочие инструкции Context -->

    <Context path="/cocoon"
              docBase="webapps/cocoon"
              debug="0"
              reloadable="true" >
    </Context>
  </ContextManager>
</Server>
```


Другими словами, запросы, основанные на URI */cocoon* (такие как */cocoon/index.xml*), должны быть отображены в контекст указанного каталога (*webapps/cocoon*). Разумеется, следует создать каталоги только что определенного контекста. Создайте каталоги *cocoon* и *cocoon/WEB-INF* в каталоге Tomcat *webapps*. Должна получиться структура каталогов, подобная представленной на рис. 10.1.

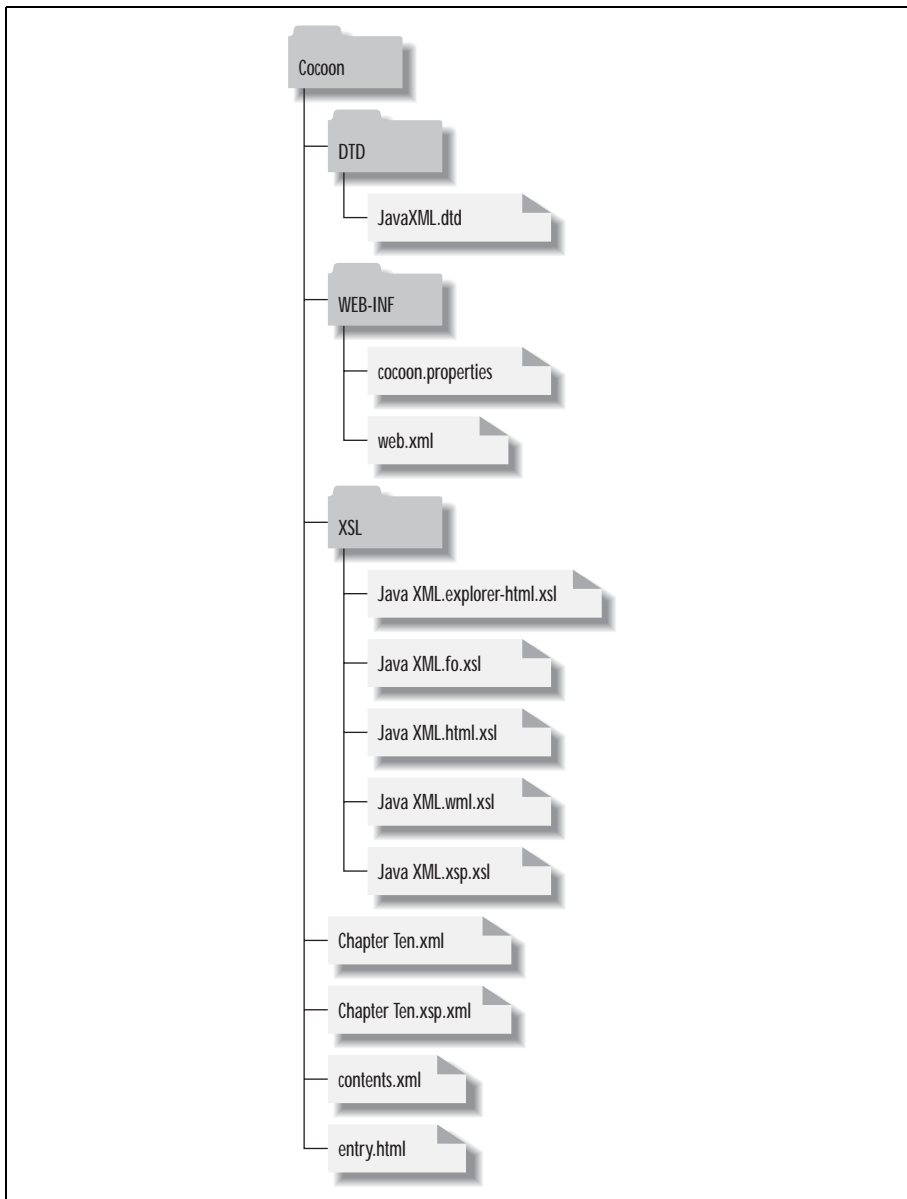


Рис. 10.1. Структура каталогов контекста Cocoon

После этого необходимо скопировать несколько файлов из дистрибутива Coooon в контекст. Скопируйте файлы *conf/cocoon.properties* и *src/WEB-INF/web.xml* в каталог *TOMCAT_HOME/webapps/cocoon/WEB-INF/*. Теперь осталось лишь изменить новую копию файла *web.xml*. Он должен ссылаться на новую копию файла *cocoon.properties*:

```
<web-app>
  <servlet>
    <servlet-name>org.apache.cocoon.Cocoon</servlet-name>
    <servlet-class>org.apache.cocoon.Cocoon</servlet-class>
    <init-param>
      <param-name>properties</param-name>
      <param-value>WEB-INF/cocoon.properties</param-value>
    </init-param>
  </servlet>

  <servlet-mapping>
    <servlet-name>org.apache.cocoon.Cocoon</servlet-name>
    <url-pattern>*.xml</url-pattern>
  </servlet-mapping>
</web-app>
```

Остался последний, довольно неприятный шаг. Tomcat автоматически загружает все *jar*-файлы из каталога *TOMCAT_HOME/lib/*, причем последовательность загрузки определяется алфавитным порядком следования имен *jar*-файлов. Проблема заключается в том, что для работы Coooon требуется реализация DOM Level 2 (из анализатора Xerces, который входит в состав Coooon в виде файла *xerces_1_2.jar*); однако Tomcat использует реализацию DOM Level 1, включенную в файл *parser.jar*. Разумеется, из-за загрузки в алфавитном порядке файл *parser.jar* загружается до файла *xerces_1_2.jar*, и Coooon остается не у дел. От этой неприятности можно избавиться. Для этого переименуйте архив *parser.jar* таким образом, чтобы он загружался после Xerces (я назвал его *z_parser.jar*). Этот шаг гарантирует, что классы DOM Level 1 по-прежнему доступны Tomcat, но классы DOM Level 2 загружаются первыми и используются системой Coooon.

Выполнив все эти шаги, проверьте работу Coooon, загрузив информационный URI Coooon, который сообщает подробности об установке Coooon: *http://[имя узла:порт]/cocoon/Cocoon.xml*. По умолчанию это будет адрес *http://localhost:8080/cocoon/Cocoon.xml*. Ваш браузер отобразит информацию, подобную приведенной на рис. 10.2.¹

Сделав это, можно размещать реальное содержимое. В описанном варианте установки все запросы документов с URI, оканчивающимися на *.xml* и принадлежащими указанному контексту, будут обрабатываться сервлетом Coooon.

¹ Разумеется, чтобы система заработала, среда исполнения сервлетов должна быть уже настроена и интегрирована с веб-сервером. – *Примеч. науч. ред.*

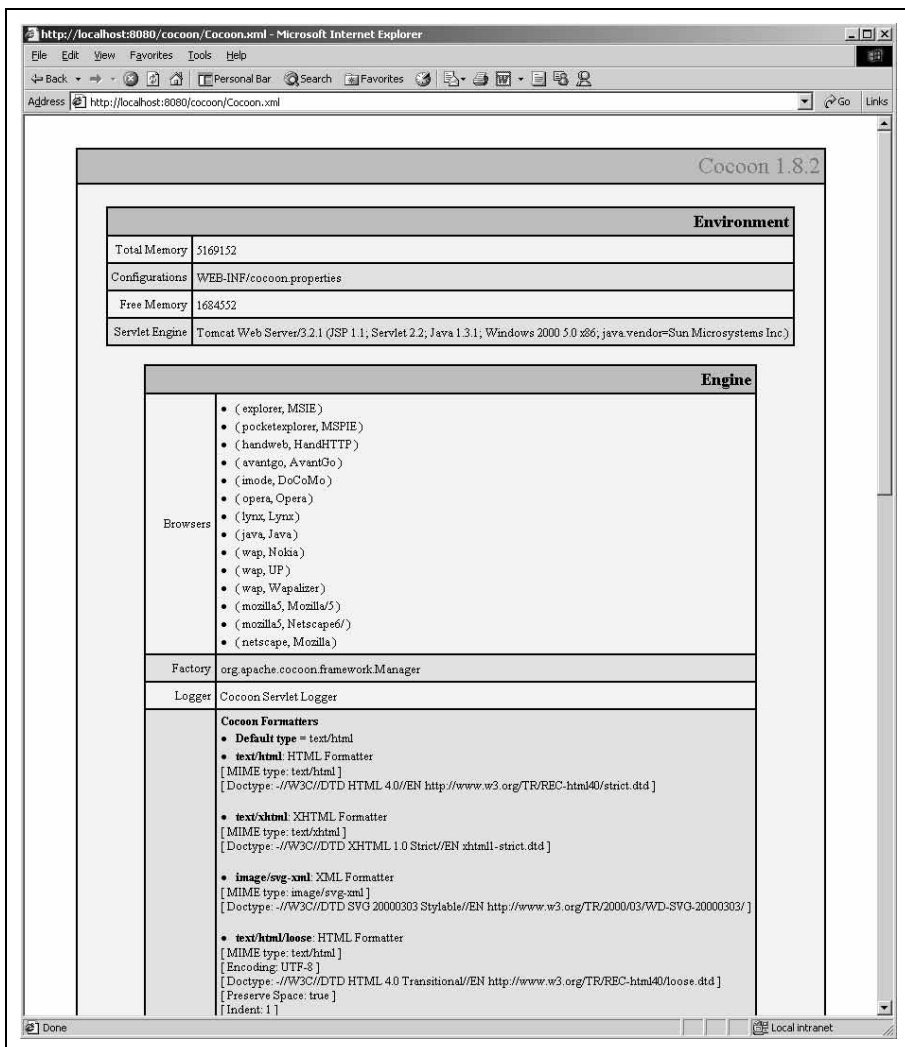


Рис. 10.2. Проверка установки Cocoon

Применение системы публикации

Использование хорошей системы публикации, такой как Cocoon, не требует каких-либо специальных знаний. Это не настолько сложное приложение, чтобы пользователям приходилось к нему привыкать. На самом деле все применения Cocoon основаны на простых URL, которые вводятся в стандартном веб-браузере. Создание динамического HTML из XML, просмотр XML-данных, преобразованных в формат PDF, и даже создание приложений VRML из XML осуществляется

простым набором в веб-браузере URL-адреса нужного XML-файла. Остальное делает система Coooon и мощь технологии XML.

Преобразование XML в HTML

Наконец, наша система публикации установлена и корректно обрабатывает запросы для адресов, оканчивающихся на *.xml*, и мы можем увидеть, как она публикует наши XML-файлы. В состав Coooon входит набор примеров XML-файлов и связанных с ними таблиц стилей XSL, который хранится в подкаталоге *samples/*. Но у нас имеются собственный XML-документ и таблица стилей XSL из предыдущих глав, поэтому попробуем преобразовать содержание книги в формате XML (*contents.xml*) с помощью таблицы стилей XSL (*JavaXML.html.xsl*) из главы 2. Найдите сохраненный XML-файл и скопируйте его в корневой каталог документов Coooon *webapps/cocoon/*. Наш документ связан с таблицей стилей *XSL/JavaXML.html.xsl*. Создайте каталог *XSL/* в корневом каталоге документов веб-сервера и скопируйте в этот каталог таблицу стилей. XML-документ также ссылается и на DTD; ссылку следует либо закомментировать, либо скопировать файл *JavaXML.dtd* (см. главу 2) в каталог *DTD/*, созданный в корневом каталоге веб-сервера.

Разместив XML-документ и таблицу стилей, можно обратиться к документу с помощью веб-браузера, указав URL *http://<имя узла>:<порт>/cocoon/contents.xml*.¹ Если все инструкции, связанные с установкой Coooon, выполнены, то преобразованный XML-документ должен выглядеть примерно так, как показано на рис. 10.3.

Это должно быть практически тривиальным. Когда Coooon установлен и настроен, выдача динамического содержимого становится простейшей задачей! Соответствие между расширениями XML и системой Coooon работает для всех запросов в контексте Coooon.

Преобразование XML в формат PDF

Говоря о визуализации данных XML, я концентрировался на преобразовании XML в HTML. Но это лишь начало разговора о форматах, в которые может быть преобразован XML. В качестве формата конечных документов может выступать любой из многочисленных языков разметки; но, кроме того, Java предоставляет библиотеки для преобразования XML в форматы, не имеющие отношения к разметке. Наиболее популярной и стабильной библиотекой такого рода является FOP, процессор форматирования, разработанный в рамках проекта Apache XML.

¹ Здесь есть некоторое противоречие. Coooon, фактически, срабатывает только при указании контекста (каталог *cocoon* в данном случае). Запрос *http://localhost/contents.xml* будет обработан только веб-сервером, и сервер, скорее всего, сообщит, что документ не найден (404). – *Примеч. науч. ред.*

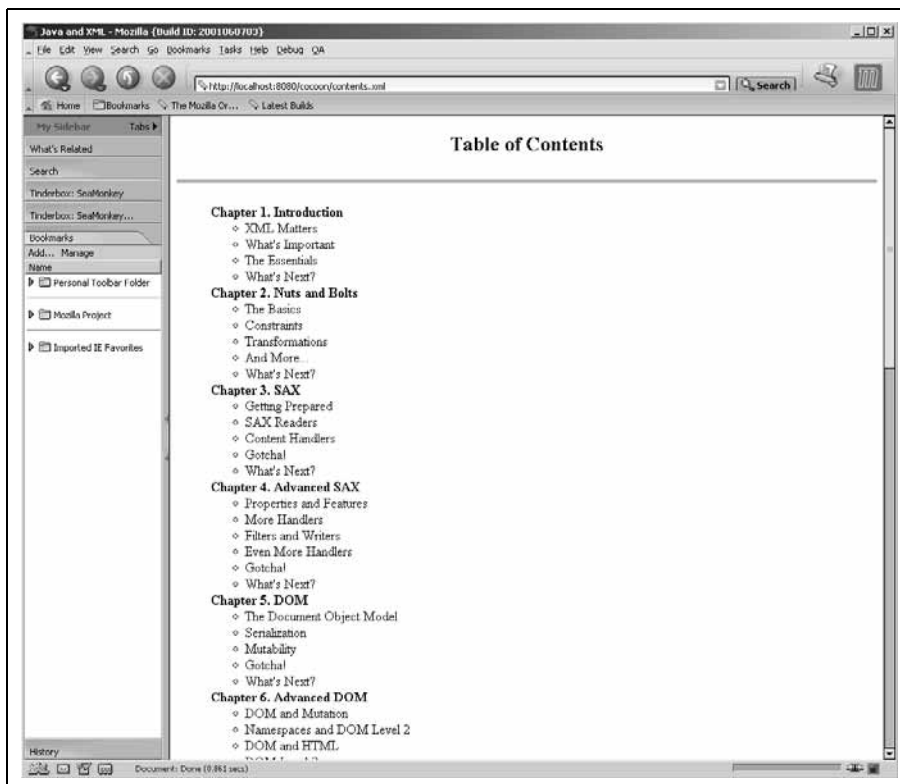


Рис. 10.3. Cocoon в действии для файла contents.xml

С помощью этого пакета любая система публикации, включая Cocoon, получает возможность преобразовывать документы XML в формат PDF, для просмотра которого обычно используют Adobe Acrobat (<http://www.adobe.com>).

Важность преобразования документа из XML-формата в формат PDF невозможно переоценить, и, если говорить о сайтах, ориентированных на публикацию отдельных документов, таких как сайты печатных изданий или издательских компаний, подобная возможность способна полностью преобразить доставку данных в Сети. Рассмотрим следующий XML-документ – отрывок из этой главы, представленный в формате XML (пример 10.1).

Пример 10.1. XML-версия «Java и XML»

```
<?xml version="1.0"?>
<cocoon-process type="xslt"?>
<xml-stylesheet href="XSL/JavaXML.fo.xsl" type="text/xsl"?>
<book>
  <cover>
```

```

<title>Java и XML</title>
  <author>Бретт Маклафлин </author>
</cover>

<contents>
  <chapter title="Системы веб-публикации" number="10">
    <paragraph> С этой главы начинается обзор специальных тем, посвященных
    Java и XML. Мы уже рассмотрели основы применения XML в Java, уделили внимание
    использованию API SAX, DOM, JDOM и JAXP для работы с XML, а также вопросам
    использования и создания XML-документов. Теперь, когда вы понимаете, как
    использовать XML в вашем коде, можно перейти к обсуждению конкретных
    приложений. В следующих шести главах представлены важнейшие приложения XML и,
    в частности, их реализации в мире технологий Java. Хотя уже существуют тысячи
    важнейших приложений XML, темы, рассматриваемые в этих главах, постоянно
    находятся в центре внимания и несут в себе значительный потенциал для
    изменения традиционных процессов разработки приложений.
    </paragraph>

    <sidebar title="Чем больше все меняется, тем больше остается неизменным">
    Те, кто читал первое издание книги, обнаружат, что большая часть разговора о
    Codox в этой главе осталась прежней. Хоть я и обещал, что к этому времени
    выйдет Codox 2, и собирался написать целую главу, посвященную ему, все
    развивалось не так быстро, как я того ожидал. Стефано Маццокки (Stefano
    Mazzocchi) – автор Codox – решил наконец закончить учебу (так держать,
    Стефано!), и в результате разработка Codox 2 замедлилась. Основная
    разработка по-прежнему сосредоточена на Codox 1.x, так что ее и следует
    использовать. Раздел, посвященный Codox 2, обновлен и отражает ожидаемые
    нововведения. В ближайшие месяцы следует ожидать новых книг от O'Reilly,
    посвященных вопросам, связанным с системой Codox. – </sidebar>

    <paragraph> Первая актуальная тема, которую я рассмотрю, связана с
    приложением XML, которое вызвало наибольший резонанс среди разработчиков на
    XML и Java. Речь идет о системе веб-публикации. Хотя я постоянно говорю, что
    визуализацию содержимого документа, возможно, переоценивают по сравнению со
    значимостью переносимости данных, которую обеспечивает XML, применение XML
    для формирования представления данных все же является очень важным.
    Потребность в визуализации возрастает, когда речь заходит о веб-
    ориентированных приложениях. </paragraph>
  </chapter>
</contents>
</book>

```

Вы уже видели, как таблицы стилей XSL позволяют преобразовать документ в HTML. Однако преобразование целой главы книги в формат HTML может привести к созданию гигантского HTML-документа и, конечно, совершенно не читаемого; возможные читатели, желающие получить книгу в режиме реального времени, в общем случае предпочитают PDF-документ. С другой стороны, статическая генерация PDF-файла из текста главы означает, что изменения в тексте главы должны быть согласованы с последующей генерацией PDF-файла. Использование XML в качестве единого формата документов означает, что главу можно легко обновить (с помощью любого редактора XML), перефор-

матировать в SGML для печати бумажной копии, передать другим компаниям и приложениям и включить в другие книги и сборники. А теперь добавьте к этому мощному набору характеристик возможность для пользователей набрать URL и получить доступ к книге в формате PDF – и вы получаете законченную систему публикации.

Здесь не рассмотрены объекты форматирования и FOP для Java в подробностях, но исчерпывающую информацию по объектам форматирования в спецификации XSL можно найти на сайте <http://www.w3.org/TR/xsl/>. Пример 10.2 представляет собой таблицу стилей XSL с применением объектов форматирования, определяющих преобразование документа XML, соответствующего данной главе, в PDF-документ.

Пример 10.2. Таблица стилей XSL для преобразования в PDF

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:fo="http://www.w3.org/1999/XSL/Format">
  <xsl:template match="book">
    <xsl:processing-instruction name="cocoon-format">
      type="text/xslfo"
    </xsl:processing-instruction>
    <fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
      <fo:layout-master-set>
        <fo:simple-page-master
          master-name="right"
          margin-top="75pt"
          margin-bottom="25pt"
          margin-left="100pt"
          margin-right="50pt">
          <fo:region-body margin-bottom="50pt"/>
          <fo:region-after extent="25pt"/>
        </fo:simple-page-master>
        <fo:simple-page-master
          master-name="left"
          margin-top="75pt"
          margin-bottom="25pt"
          margin-left="50pt"
          margin-right="100pt">
          <fo:region-body margin-bottom="50pt"/>
          <fo:region-after extent="25pt"/>
        </fo:simple-page-master>
        <fo:page-sequence-master master-name="psmOddEven">
          <fo:repeatable-page-master-alternatives>
            <fo:conditional-page-master-reference
              master-name="right"
              page-position="first"/>
            <fo:conditional-page-master-reference
              master-name="right"
              odd-or-even="even"/>
          <fo:conditional-page-master-reference
```

```

        master-name="left"
        odd-or-even="odd"/>
        <!-- recommended fallback procedure -->
        <fo:conditional-page-master-reference
            master-name="right"/>
    </fo:repeatable-page-master-alternatives>
</fo:page-sequence-master>
</fo:layout-master-set>

<fo:page-sequence master-name="psmOddEven">
    <fo:static-content flow-name="xsl-region-after">
        <fo:block text-align-last="center" font-size="10pt">
            <fo:page-number/>
        </fo:block>
    </fo:static-content>
    <fo:flow flow-name="xsl-region-body">
        <xsl:apply-templates/>
    </fo:flow>
</fo:page-sequence>
</fo:root>
</xsl:template>

<xsl:template match="cover">
    <fo:block font-size="10pt"
        space-before.optimum="10pt">
        <xsl:value-of select="title"/>
        (<xsl:value-of select="author"/>)
    </fo:block>
</xsl:template>

<xsl:template match="contents">
    <xsl:apply-templates/>
</xsl:template>

<xsl:template match="chapter">
    <fo:block font-size="24pt"
        text-align-last="center"
        space-before.optimum="24pt">
        <xsl:value-of select="@number" />.
        <xsl:value-of select="@title" />
        <xsl:apply-templates/>
    </fo:block>
</xsl:template>

<xsl:template match="paragraph">
    <fo:block font-size="12pt"
        space-before.optimum="12pt"
        text-align="justify">
        <xsl:apply-templates/>
    </fo:block>
</xsl:template>

<xsl:template match="sidebar">
    <fo:block font-size="14pt"

```



```

        font-style="italic"
        color="blue"
        space-before.optimum="16pt"
        text-align="center">
    <xsl:value-of select="@title" />
</fo:block>
<fo:block font-size="12pt"
        color="blue"
        space-before.optimum="16pt"
        text-align="justify">
    <xsl:apply-templates/>
</fo:block>
</xsl:template>
</xsl:stylesheet>

```

Если создать оба этих файла, сохранить главу в файле *chapterTen.xml*, а таблицу стилей XSL в файле *JavaXML.fo.xsl* в подкаталоге *XSL/*, то результат преобразования можно получить с помощью веб-браузера. Убедитесь, что располагаете программой Adobe Acrobat Reader и соответствующим модулем (plug-in) для веб-браузера, а затем обратитесь к только что созданному XML-документу. Результаты показаны на рис. 10.4.

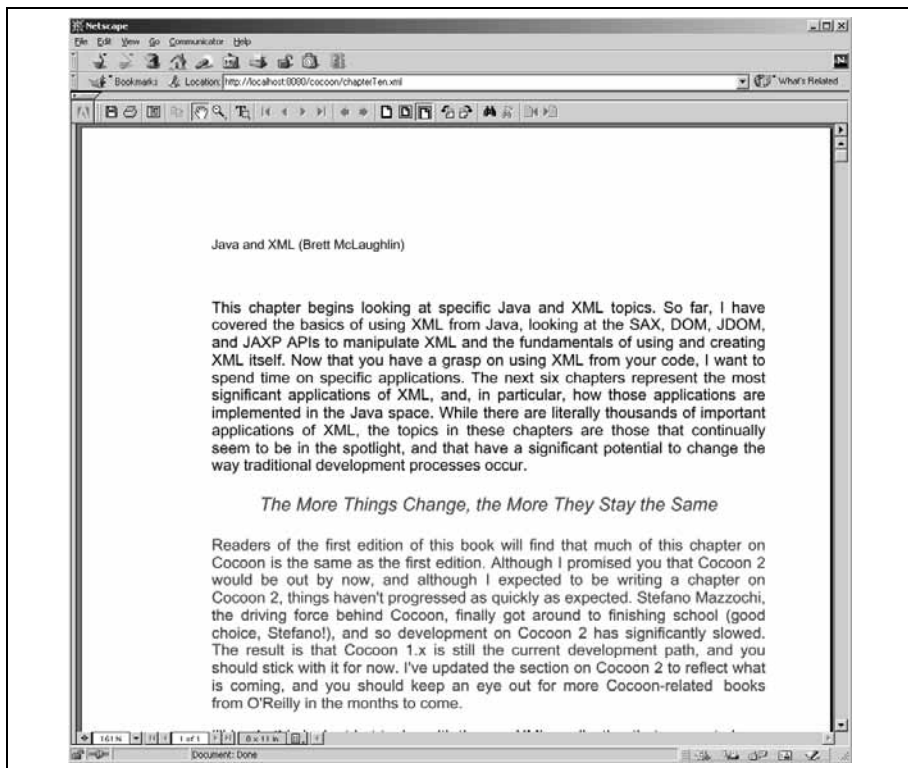


Рис. 10.4. Результат преобразования в PDF из *chapterTen.xml*

Форматирование в зависимости от браузера

Помимо специальных типов преобразований, таких как конвертирование в PDF, система Socoop предусматривает динамическую обработку, основанную на способе доставки информации. Распространенным примером такой обработки является использование различного форматирования для разных типов клиентов. В традиционном веб-окружении это позволяет различными способами визуализировать XML-документы исходя из категории и версии браузера клиента. Для клиента, работающего с Internet Explorer, может использоваться иное представление данных, чем для клиента, работающего с Netscape. Принимая во внимание недавние «войны» версий HTML, DHTML и JavaScript, в которых активно участвовали компании Netscape и Microsoft, эта мощная возможность стоит того, чтобы иметь ее в своем распоряжении. Система Socoop обеспечивает встроенную поддержку для множества распространенных типов браузеров. Найдите файл *socoop.properties*, о котором мы говорили ранее, откройте его и перейдите в конец файла. Вы увидите следующую секцию (в более новых версиях она может несколько отличаться):

```
#####
# User Agents (Browsers) #
#####

# NOTE: numbers indicate the search order. This is very important since
# some words may be found in more than one browser description. (MSIE is
# presented as "Mozilla/4.0 (Compatible; MSIE 4.01; ...")
#
# for example, the "explorer=MSIE" tag indicates that the XSL stylesheet
# associated to the media type "explorer" should be mapped to those
# browsers that have the string "MSIE" in their "user-Agent" HTTP header.
[Перевод:
#####
# Пользовательские агенты (браузеры) #
#####

# ПРИМЕЧАНИЕ: числа определяют порядок поиска. Это ОЧЕНЬ ВАЖНО,
# поскольку некоторые слова могут быть найдены в описании различных
# браузеров.
# (MSIE заявляет о себе как "Mozilla/4.0 (Compatible; MSIE 4.01; ...")
#
# например, метка "explorer=MSIE" указывает на то, что таблица стилей XSL,
# связанная с типом клиента "explorer", должна быть привязана к тем
# браузерам, которые
# имеют строку "MSIE" в HTTP-заголовке "user-Agent".]

browser.0 = explorer=MSIE
browser.1 = pocketexplorer=MSPIE
```

```

browser.2 = handweb=HandHTTP
browser.3 = avantgo=AvantGo
browser.4 = imode=DoCoMo
browser.5 = opera=Opera
browser.6 = lynx=Lynx
browser.7 = java=Java
browser.8 = wap=Nokia
browser.9 = wap=UP
browser.10 = wap=Wapalizer
browser.11 = mozilla5=Mozilla/5
browser.12 = mozilla5=Netscape6/
browser.13 = netscape=Mozilla
    
```

Здесь нас интересуют ключевые слова после первого знака равенства. Например, `explorer`, `lynx`, `java` и `mozilla5` позволяют различать агенты пользователей и являются «кодами», которые посылают браузеры при выполнении запросов по URL-адресам. В качестве примера применения таблиц стилей, основанных на этом свойстве, создадим таблицу стилей XSL, применяемую в том случае, когда клиент обращается к оглавлению книги в формате XML (*contents.xml*) с помощью браузера Internet Explorer. Скопируйте первоначальную таблицу стилей *JavaXML.html.xsl* в файл *JavaXML.explorer.html.xsl*, а затем внесите изменения, как показано в примере 10.3.

Пример 10.3. Измененная таблица стилей XSL для Internet Explorer

```

<?xml version="1.0"?>

<xsl:stylesheet xmlns:javaxml2="http://www.oreilly.com/javaxml2"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:ora="http://www.oreilly.com"
  version="1.0"
>

  <xsl:template match="javaxml2:book">
    <xsl:processing-instruction name="cocoon-format"
      type="text/html"
    </xsl:processing-instruction>
    <html>
      <head>
        <title>
          <xsl:value-of select="javaxml2:title" /> (Версия для браузера
            Explorer)
        </title>
      </head>
      <body>
        <xsl:apply-templates select="*[not(self::javaxml2:title)]" />
      </body>
    </html>
  </xsl:template>
    
```

```

<xsl:template match="javaxml2:contents">
  <center>
    <h2>Оглавление (Версия для броузера Explorer)</h2>
    <small>
      Попробуйте браузер <a href="http://www.mozilla.org">Mozilla</a>!
    </small>
  </center>
  <!-- Прочие инструкции XSL -->
</xsl:template>

<!-- Прочие шаблоны/сопоставления XSL -->

</xsl:stylesheet>

```

И хотя данный пример тривиален, у нас есть потенциальная возможность включить динамический HTML для броузера Internet Explorer 5.5, а для Netscape Navigator или Mozilla, которые реализуют менее полную поддержку DHTML, использовать стандартный HTML. При этом следует сообщить XML-документу о том, что если способ доставки (или идентификатор пользовательского агента) совпадает с определенным в файле свойств типом `explorer`, необходимо использовать соответствующую таблицу стилей XSL. Эту задачу решает дополнительная инструкция обработки, показанная в примере 10.4, и ее можно добавить в файл `contents.xml`.

Пример 10.4. Измененный документ `contents.xml`, распознавание способа доставки

```

<?xml version="1.0"?>
<!DOCTYPE Book SYSTEM "DTD/JavaXML.dtd">
<?xml-stylesheet href="XSL/JavaXML.html.xsl" type="text/xsl"?>
<?xml-stylesheet href="XSL/JavaXML.explorer-html.xsl" type="text/xsl"
  media="explorer"?>

<?cocoon-process type="xslt"?>

<!-- "Java и XML", Оглавление -->
<book xmlns="http://www.oreilly.com/javaxml2"
  xmlns:ora="http://www.oreilly.com"
  >
  <!-- XML-данные -->
</book>

```

Обращение к XML-документу при помощи броузера Netscape дает те же результаты, что и раньше. Однако если обратиться к этой странице с помощью броузера Internet Explorer, можно увидеть, что документ был преобразован с применением другой таблицы стилей и выглядит так, как показано на рис. 10.5.

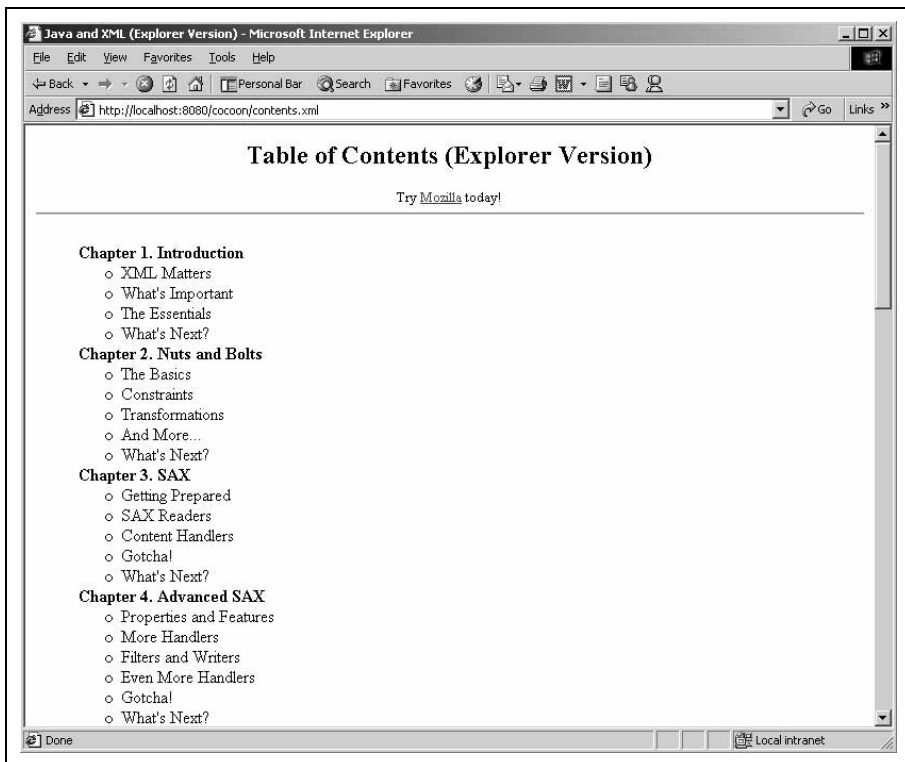


Рис. 10.5. Файл contents.xml, полученный с помощью Internet Explorer

WAP и XML

Одна из самых сильных сторон динамического применения таблиц стилей связана с использованием беспроводных устройств. Помните файл свойств?

```
#####
# User Agents (Browsers) #
#####

# NOTE: numbers indicate the search order. This is very important since
# some words may be found in more than one browser description. (MSIE is
# presented as "Mozilla/4.0 (Compatible; MSIE 4.01; ...)")
#
# for example, the "explorer=MSIE" tag indicates that the XSL stylesheet
# associated to the media type "explorer" should be mapped to those
# browsers that have the string "MSIE" in their "user-Agent" HTTP header.

browser.0 = explorer=MSIE
browser.1 = pocketexplorer=MSPIE
browser.2 = handweb=HandHTTP
```

```

browser.3 = avantgo=AvantGo
browser.4 = imode=DoCoMo
browser.5 = opera=Opera
browser.6 = lynx=Lynx
browser.7 = java=Java
browser.8 = wap=Nokia
browser.9 = wap=UP
browser.10 = wap=Wapalizer
browser.11 = mozilla5=Mozilla/5
browser.12 = mozilla5=Netscape6/
browser.13 = netscape=Mozilla

```

Выделенные записи указывают, что для доступа к содержимому документа используется беспроводное устройство, такое как телефон, имеющий выход в Интернет. Система Coccoon в состоянии определить тип броузера, запросившего документ (Internet Explorer или Netscape), и вернуть результат, отформатированный с применением соответствующей таблицы стилей; и одна из этих таблиц стилей может предназначаться для WAP-устройств. Добавьте ссылку на еще одну таблицу стилей в документ *contents.xml*:

```

<?xml version="1.0"?>
<!DOCTYPE Book SYSTEM "DTD/JavaXML.dtd">
<?xml-stylesheet href="XSL/JavaXML.html.xml" type="text/xml"?>
<?xml-stylesheet href="XSL/JavaXML.explorer-html.xml" type="text/xml"
    media="explorer"?>
<?xml-stylesheet href="XSL/JavaXML.wml.xml" type="text/xml"
    media="wap"?>

<?cocoon-process type="xslt"?>

<!-- "Java и XML", Оглавление -->
<book xmlns="http://www.oreilly.com/javaxml2"
    xmlns:ora="http://www.oreilly.com"
>
    <!-- Оглавление в формате XML -->
</book>

```

Теперь необходимо создать эту новую таблицу стилей для WAP-устройств. Как правило, при создании таблицы стилей для устройства WAP используется язык разметки для беспроводных устройств (Wireless Markup Language, WML). Он является вариантом HTML, но имеет несколько иной метод представления различных страниц. Когда беспроводное устройство запрашивает документ по URL, возвращаемый ответ должен размещаться внутри элемента *wml*. Корневой элемент может содержать несколько *карт (cards)*, каждая из которых определяется при помощи элемента WML *card*. Устройство загружает сразу несколько карт (обычно этот набор называется *колодой (deck)*) таким образом, что ему не требуется обращаться к серверу за дополнительными страницами. В примере 10.5 показана простая WML-страница, использующая упомянутые конструкции.

Пример 10.5. Простая WML-страница

```
<wml>
<card id="index" title="Домашняя страница">
  <p align="left">
    <i>Основное меню</i><br />
    <a href="#title">Титульная страница</a><br />
    <a href="#myPage">Моя страница</a><br />
  </p>
</card>

<card id="title" title="Моя титульная страница ">
  Добро пожаловать на мою титульную страницу!<br />
  Я очень рад вас видеть.
</card>

<card id="myPage" title="Здравствуй, мир ">
  <p align="center">
    Здравствуй, мир!
  </p>
</card>
</wml>
```

Запрос этого документа приводит к получению меню и пары страниц, доступ к которым можно получить с помощью пунктов меню. Полная спецификация WML 1.1, а также иные спецификации, связанные с WAP, доступны в Интернете на странице http://www.wapforum.org/what/technical_1_1.htm. Также рекомендую книгу Мартина Фроста (Martin Frost) «Learning WML and WMLScript», O'Reilly (Изучаем WML и WMLScript). Кроме того, можно загрузить WAP-версию продукта Openwave™ SDK с сайта http://www.openwave.com/products/developer_products/sdk/. Этот программный продукт предназначен для разработки WAP-страниц и в числе прочих инструментов включает эмуляцию беспроводного устройства.¹ С его помощью можно разработать таблицу стилей XSL для выдачи WML-кода устройствам WAP и проверить результаты, указав в эмуляторе SDK адрес <http://<имя узла>:<порт>/cocoon/contents.xml>.

Поскольку телефонные дисплеи гораздо меньше, чем экраны компьютеров, мы хотим отобразить лишь некоторую часть информации из оглавления книги в формате XML. В примере 10.6 приведена таблица стилей XSL, которая выдает три карты WML. Первая из них представляет собой меню, имеющее ссылки на две другие карты. Вторая карта генерирует оглавление из документа *contents.xml*. Третья карта – это просто страничка с информацией о правообладании. Данную таблицу

¹ *phone.com* была поглощена компанией *openwave.com*, и обнаружить в числе продуктов Openwave UP.SDK не удалось. Зато нашелся Openwave SDK, WAP Edition, который выполняет точно те же функции. – *Примеч. науч. ред.*

стилей можно сохранить под именем *JavaXML.wml.xsl* в подкаталоге *XSL/* корневого каталога документов веб-сервера.

Пример 10.6. Таблица стилей для вывода WML

```
<?xml version="1.0"?>

<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:javaxml2="http://www.oreilly.com/javaxml2"
  xmlns:ora="http://www.oreilly.com"
  exclude-result-prefixes="javaxml2 ora"
>

<xsl:template match="javaxml2:book">
  <xsl:processing-instruction name="cocoon-format"
    type="text/wml"
  </xsl:processing-instruction>

  <wml>
    <card id="index" title="{javaxml2:title}">
      <p align="center">
        <i><xsl:value-of select="javaxml2:title"/></i><br />
        <a href="#contents">Contents</a><br/>
        <a href="#copyright">Copyright</a><br/>
      </p>
    </card>

    <xsl:apply-templates select="javaxml2:contents" />

    <card id="copyright" title="Copyright">
      <p align="center">
        Copyright 2000, O&apos;Reilly & Associates
      </p>
    </card>
  </wml>
</xsl:template>

<xsl:template match="javaxml2:contents">
  <card id="contents" title="Contents">
    <p align="center">
      <i>Contents</i><br />
      <xsl:for-each select="javaxml2:chapter">
        <xsl:value-of select="@number" />.
        <xsl:value-of select="@title" /><br />
      </xsl:for-each>
    </p>
  </card>
</xsl:template>

</xsl:stylesheet>
```


Если не считать тегов WML, большая часть этого примера должна выглядеть привычно. Здесь также присутствует инструкция обработки для Coooon, целевое приложение которой – coocon-format. Передаваемые инструкцией данные (type="text/wml") предписывают системе Coooon осуществлять вывод этой таблицы стилей с заголовком содержимого text/wml (вместо обычного text/html или text/plain). Очень важен атрибут, добавленный к корневому элементу таблицы стилей:

```
<?xml version="1.0"?>

<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:javaxml2="http://www.oreilly.com/javaxml2"
    xmlns:ora="http://www.oreilly.com"
    exclude-result-prefixes="javaxml2 ora"
>
```

По умолчанию любые объявления пространств имен XML, за исключением объявления пространства имен XSL, добавляются в корневой элемент результата преобразований. В данном примере к корневому элементу результата преобразования wml были бы добавлены объявления пространств имен, связанных с префиксами javaxml2 и ora:

```
<wml xmlns:javaxml2="http://www.oreilly.com/javaxml2"
    xmlns:ora="http://www.oreilly.com"
>
  <!-- Данные WML -->
</wml>
```

Но такое добавление вызвало бы сообщение об ошибке в WAP-браузере, поскольку xmlns:javaxml2 и xmlns:ora не являются допустимыми атрибутами элемента wml. WAP-браузеры не столь снисходительны, как HTML-браузеры, и оставшаяся часть WML-данных не будет показана. Тем не менее это пространство должно быть объявлено, чтобы таблица стилей XSL могла выполнять сопоставление шаблонов для исходного документа, который использует пространство имен, связанное с префиксом javaxml. Для решения этой проблемы в XSL предусмотрен атрибут exclude-result-prefixes, который следует добавить к элементу xsl:stylesheet. Префикс пространства имен, указанный в этом атрибуте, не будет добавляться к результату преобразований, а именно это нам и нужно. Теперь результат будет выглядеть следующим образом:

```
<wml>
  <!-- Данные WML -->
</wml>
```

Этот код отлично воспринимается WAP-браузером. Если у вас эмулятор WAP-устройства (Openwave SDK) загружен, можно обратиться с его помощью к оглавлению в формате XML и увидеть результаты.

На рис. 10.6 показано главное меню, полученное в результате преобразования с использованием таблицы стилей для WML. Преобразование выполняется в момент, когда WAP-устройство запрашивает файл *contents.xml* через систему публикации Coccoon.



Рис. 10.6. Основное меню для оглавления «Java и XML»

Внимание

В тестируемых автором версиях браузера UP.SDK ссылка на сущность `OreillyCopyright` не разрешалась. Чтобы примеры работали, пришлось закомментировать эту строку в XML-коде. Вероятно, вам потребуется делать то же самое до тех пор, пока эта ошибка в эмуляторе не будет исправлена.

На рис. 10.7 показано сгенерированное оглавление книги, доступ к которому был осуществлен посредством нажатия кнопки `Link`, когда на дисплее указана ссылка *Contents*.



Рис. 10.7. Оглавление в формате WML

За более подробной информацией о WML и WAP обратитесь к страницам в Интернете: <http://www.openwave.com>¹ и <http://www.wapforum.org>. На каждом из этих сайтов представлен богатый выбор ресурсов, связанных с разработкой приложений для мобильных устройств.

К этому моменту читатели должны уже хорошо представлять все разнообразие конечных документов, которые можно создать с помощью системы Cocoon. С минимальным количеством затраченных усилий и дополнительных таблиц стилей один и тот же XML-документ может быть представлен во множестве форматов для самых разнообразных

¹ Внимательные читатели заметят отсутствие ссылок на [phone.com](http://www.phone.com). Дело в том, что [phone.com](http://www.phone.com) теперь стал частью OpenWave (<http://www.openwave.com>).

типов клиентов. Это одна из возможностей, делающих системы веб-публикации столь мощным средством. Без XML и подобных систем для каждого типа клиента пришлось бы создавать отдельный сайт. Теперь, когда вы убедились в гибкости системы Cocoon в вопросах создания выходных данных, перейдем к обзору технологии Cocoon, которая позволяет динамически создавать и модифицировать исходные данные для этих преобразований.

XSP

Аббревиатура XSP означает *расширяемые серверные страницы* (Extensible Server Pages), и эта технология представляет собой, возможно, наиболее важную разработку, вышедшую из недр проекта Cocoon. Серверные страницы Java (JSP) позволяют внедрять метки и код Java в обычные HTML-страницы. Когда поступает запрос на получение такой страницы JSP, внедренный код выполняется, а результаты вставляются в передаваемый клиенту HTML-код¹. Эта технология стала широко применяться разработчиками на Java и ASP, упрощая, якобы, серверное программирование на Java и позволяя разделять логику и выводимые данные. Однако здесь все еще существует несколько серьезных проблем. Во-первых, JSP в действительности не обеспечивает разделения содержания и представления. Проблема все та же, и мы о ней уже говорили: изменение баннера, цвета или размера шрифта требует внесения изменений в JSP (включая внедренный код на Java и ссылки на компоненты JavaBean). JSP привязывает содержимое (чистые данные) к представлению точно так же, как это делает статический HTML. Во-вторых, JSP-страницы не могут быть преобразованы в другие форматы и не могут использоваться в нескольких приложениях, поскольку спецификация JSP разработана главным образом «вокруг» выдачи данных.

XSP решает эти проблемы. По сути дела, XSP – это просто XML. Рассмотрим XSP-страницу, приведенную в примере 10.7.

Пример 10.7. Простая XSP-страница

```
<?xml version="1.0"?>
<?cocoon-process type="xsp"?>
<?cocoon-process type="xslt"?>
<?xml-stylesheet href="myStylesheet.xsl" type="text/xsl"?>
<xsp:page language="java"
```

¹ Это чересчур сильное упрощение. В действительности JSP прекомпилируется в сервлет, и выводом данных управляет класс `PrintWriter`. Дополнительную информацию о JSP можно найти в книге Ханса Бергстена (Hans Bergsten) «JavaServer Pages», вышедшей в издательстве O'Reilly & Associates.

```

        xmlns:xsp="http://www.apache.org/1999/XSP/Core"
    >

    <xsp:logic>
        private static int numHits = 0;

        private synchronized int getNumHits() {
            return ++numHits;
        }
    </xsp:logic>

    <page>
        <title>Счетчик посещений</title>

        <p>Эта страница была запрошена <xsp:expr>getNumHits()</xsp:expr> раз.</p>
    </page>
</xsp:page>

```

Этот пример следует всем соглашениям XML. Считайте пока, что доступ к содержимому элемента `xsp:logic` для анализатора XML «запрещен», – позже мы к этому вернемся. В остальном документ представляет собой обычный XML-документ с несколькими новыми элементами. Он ссылается на таблицу стилей XSL, в которой нет ничего особенного, как видно из примера 10.8.

Пример 10.8. Таблица стилей XSL для XSP-страницы

```

<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
        xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    >

    <xsl:template match="page">
        <xsl:processing-instruction name="cocoon-format">
            type="text/html"
        </xsl:processing-instruction>
        <html>
            <head>
                <title><xsl:value-of select="title"/></title>
            </head>
            <body>
                <xsl:apply-templates select="*[not(self::title)]" />
            </body>
        </html>
    </xsl:template>

    <xsl:template match="p">
        <p align="center">
            <xsl:apply-templates />
        </p>
    </xsl:template>
</xsl:stylesheet>

```

Таким образом, XSP легко решает первую главную проблему JSP, обеспечивая четкое разделение содержимого и представления. Такое разделение дает возможность разработчикам управлять созданием содержимого документов (XSP-страницу можно сгенерировать с помощью сервлета или другого кода на Java), в то время как авторы XML- и XSL-документов могут управлять представлением и форматированием данных, изменяя таблицы стилей XSL. Так же легко XSP восполняет другой существенный недостаток JSP: поскольку обработка XSP выполняется до применения каких-либо таблиц стилей, получаемый XML-документ может быть преобразован в любой другой формат. XSP сохраняет все преимущества XML. XSP-страницу можно передавать между приложениями или использовать для представления данных.

Создание XSP-страницы

Теперь, когда вы познакомились с технологией XSP, можно создать собственную XSP-страницу. Обратимся для этого к созданному ранее XML-документу, представляющему собой часть этой главы и ранее уже преобразованному в формат PDF. Вместо того чтобы просто использовать этот документ для вывода информации, предположим, что автор хочет предоставить возможность редактору просматривать документ в процессе написания. Однако в дополнение к тексту книги редактор должен иметь возможность видеть примечания автора, которые все остальные видеть не должны: например, замечания о стиле и форматировании. Прежде всего, добавим примечание к созданному ранее файлу *chapterTen.xml*:

```
<?xml version="1.0"?>
<?cocoon-process type="xslt"?>
<?xml-stylesheet href="XSL/JavaXML.fo.xsl" type="text/xsl"?>
<book>
  <cover>
    <title>Java и XML</title>
    <author>Бретт Мак-Лахлин </author>
  </cover>
  <contents>
    <chapter title="Системы веб-публикации" number="10">
```

```
      <paragraph> С этой главы начинается обзор специальных тем, посвященных
Java и XML. Мы уже рассмотрели основы применения XML в Java, уделили внимание
использованию API SAX, DOM, JDOM и JAXP для работы с XML, а также вопросам
использования и создания XML-документов. Теперь, когда вы понимаете, как
обеспечить взаимодействие XML со своим кодом, можно перейти к обсуждению
конкретных приложений. В следующих шести главах представлены важнейшие
приложения XML и, в частности, их реализации в мире технологий Java. Хотя уже
существуют тысячи важнейших приложений XML, темы, рассматриваемые в этих
главах, постоянно находятся в центре внимания и несут в себе значительный
потенциал для изменения традиционных процессов разработки приложений.
```

```
</paragraph>
```

```
<authorComment>Майк, ты не считаешь, что следующая врезка несколько велика? Я вполне могу ее выбросить, если и без нее все понятно.
```

```
</authorComment>
```

```
<sidebar title="Чем больше все меняется, тем больше остается неизменным">
Те, кто читал первое издание книги, обнаружат, что большая часть информации о Сосооп в этой главе осталась прежней. Хотя я и обещал, что к этому времени выйдет Сосооп 2, и собирался написать целую главу, посвященную ему, все развивалось не так быстро, как я того ожидал. Стефано Маццокки (Stefano Mazzocchi) – автор Сосооп – решил наконец закончить учебу (так держать, Стефано!), и в результате разработка Сосооп 2 замедлилась. Основная разработка по-прежнему сосредоточена на Сосооп 1.x, так что ее и следует использовать. Раздел, посвященный Сосооп 2, обновлен и отражает ожидаемые нововведения. В ближайшие месяцы следует ожидать новых книг от O’Reilly, посвященных системе Сосооп. – </sidebar>
```

```
<paragraph>Первая актуальная тема, которую я рассмотрю, связана с приложением XML, которое вызвало наибольший резонанс среди разработчиков на XML и Java. Речь идет о системе веб-публикации. Хотя я постоянно говорю, что визуализацию содержимого документа, возможно, переоценивают по сравнению со значимостью переносимости данных, которую обеспечивает XML, применение XML для формирования представления данных все же очень важно. Потребность в визуализации возрастает, когда речь заходит о веб-ориентированных приложениях. </paragraph>
```

```
</chapter>
```

```
</contents>
```

```
</book>
```

Добавив примечание в XML-документ, вставим соответствующий элемент в таблицу стилей XSL *JavaXML fo.xsl*:

```
<xsl:template match="sidebar">
  <fo:block font-size="14pt"
    font-style="italic"
    color="blue"
    space-before.optimum="16pt"
    text-align="center">
    <xsl:value-of select="@title" />
  </fo:block>
  <fo:block font-size="12pt"
    color="blue"
    space-before.optimum="16pt"
    text-align="justify">
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>

<xsl:template match="authorComment">
  <fo:block font-size="10pt"
    font-style="italic"
```

```

        color="red"
        space-before.optimum="12pt"
        text-align="justify">
    <xsl:apply-templates/>
</fo:block>
</xsl:template>

```

Примечания будут отображены чуть более мелким курсивным шрифтом красного цвета. Теперь можно преобразовать XML-документ в XSP-страницу (как показано в примере 10.9), добавив инструкции обработки для Cooon и заключив элементы в новый корневой элемент, `xsp:page`.

Пример 10.9. Преобразование документа `chapterTen.xml` в XSP-страницу

```

<?xml version="1.0"?>

<?cooon-process type="xsp"?>
<?cooon-process type="xslt"?>
<?xml-stylesheet href="XSL/JavaXML.fo.xsl" type="text/xsl"?>

<xsp:page language="java"
    xmlns:xsp="http://www.apache.org/1999/XSP/Core"
>
<book>
  <cover>
    <title>Java и XML</title>
    <author>Бретт Мак-Лаклин</author>
  </cover>

  <contents>
    <chapter title="Системы веб-публикации" number="10">
      <!-- Текст главы -->
    </chapter>
  </contents>
</book>
</xsp:page>

```

Прежде чем добавить XSP-логику, позволяющую определять, надо ли показывать примечание, создадим простую HTML-страницу, увидев которую, пользователь сможет понять, является ли он редактором книги. В реальном приложении это могла бы быть страница, осуществляющая аутентификацию и определяющая полномочия пользователя. В нашем примере она позволяет выбрать, является ли пользователь автором, редактором или только любопытствующим читателем, а также ввести пароль для проверки. Выполняющая эту задачу HTML-страница показана в примере 10.10. Сохраните этот файл под именем `entry.html` в корневом каталоге документов контекста.

Пример 10.10. Начальная страница для XSP-страницы `chapterTen.xml`

```

<html>
<head>

```



```

<title>Welcome to the Java and XML Book in Progress</title>
</head>

<body>
<h1 align="center">Процесс написания книги <i>Java и XML</i></h1>
<center>
<form action="/cocoon/chapterTen.xml" method="POST">
  Выберите свою роль:
  <select name="userRole">
    <option value="author">Я - автор </option>
    <option value="editor">Я - редактор</option>
    <option value="reader">Я - читатель</option>
  </select>
  <br />
  Введите ваш пароль:
  <input type="password" name="password" size="8" />
  <br /><br />
  <input type="submit" value="Go!" />
</form>
</center>
</body>
</html>

```

Заметьте также, что данные из HTML-формы отправляются непосредственно XSP-странице. В этом примере XSP-страница работает как сервлет. Она считывает параметры запроса, определяет, какую роль выбрал пользователь, производит проверку подлинности, используя указанный пароль, и в заключение определяет, нужно ли показывать комментарии. Для начала определим логическую переменную – она будет содержать результат сравнения параметров запроса и позволит определить, является ли пользователь автором или редактором и правильный ли пароль введен. Если значение переменной истинно, мы отображаем элемент `authorComment`. Заклучим `authorComment` в инструкции XSP, как показано ниже:

```

<xsp:logic>
  boolean authorOrEditor = false;

  // Выясняем, является ли пользователь автором или редактором
  if (authorOrEditor) {
    <xsp:content>
      <authorComment>Mike - Do you think the following sidebar is
      a little much? I could easily leave it out if it's still
      clear without it.</authorComment>
    </xsp:content>
  }
</xsp:logic>

```

Этот фрагмент должен казаться вам вполне естественным: если не принимать в расчет теги, специфичные для XSP, мы всего лишь определяем переменную и проверяем ее значение. Если значение перемен-

ной истинно, элемент `authorComment` добавляется в результат, выдаваемый XSP-страницей, в противном случае этот элемент не включается. Интересно отметить, что собственно конечный XML-документ помещается внутрь блока `xsp:logic`, в элемент `xsp:content` (который, в свою очередь, содержится во внешнем элементе `xsp:page`). Это гарантирует, что процессор XSP не будет пытаться интерпретировать никакие элементы или текст внутри блока как структуры XSP. Тот же самый код в JSP мог бы выглядеть примерно так:

```
<%
  if (authorOrEditor) {
%>
    <authorComment> Майк, ты не считаешь, что следующая врезка несколько
      велика? Я вполне могу ее выбросить, если и без нее все понятно.
    </authorComment>
<%
  }
%>
```

Этот код не является достаточно структурированным, поскольку блок JSP заканчивается перед тем, как начинается элемент `authorComment`. После элемента добавляется новый блок, который закрывает скобку, открытую в предыдущем блоке JSP. Здесь очень легко перепутать структуры кода или забыть добавить соответствующие блоки JSP. Принципы XSP требуют, чтобы каждый открытый элемент был закрыт (стандартная корректность данных XML) и чтобы один блок кода соответствовал одному элементу.

После того как логические структуры подготовлены, XSP-странице остается лишь интерпретировать параметры запроса. Можно использовать встроенную переменную XSP `request`, имитирующую объект класса `javax.servlet.http.HttpServletRequest`. Следующие дополнения к коду читают значения параметров запроса `userRole` и `password` (если они существуют). Затем значение параметра `userRole` сравнивается с типами пользователей, которые имеют право просматривать комментарии («`author`» и «`editor`»). Если соответствие имеет место, проверяется также и пароль. Если пароль совпадает с ключом для указанной роли, логическая переменная устанавливается в значение «истина», и элемент `authorComments` становится частью конечного XML-документа:

```
<xsp:logic>
  boolean authorOrEditor = false;

  // Проверяем, является пользователь автором или редактором
  <![CDATA[
String[] roleValues = request.getParameterValues("userRole");
String[] passwordValues = request.getParameterValues("password");
if ((roleValues != null) && (passwordValues != null)) {
  String userRole = roleValues[0];
  String password = passwordValues[0];
```

```

    if (userRole.equals("author") && password.equals("brett")) {
        authorOrEditor = true;
    } else
        if (userRole.equals("editor") && password.equals("mike")) {
            authorOrEditor = true;
        }
    }
]]>

    if (authorOrEditor) {
...

```

Обратите внимание, что приличный по размеру фрагмент кода содержится внутри секции CDATA. Помните, что страницы XSP обрабатываются как документы XML, и должны следовать всем правилам данных XML. Но двойные кавычки и амперсанды, используемые во фрагментах кода Java, недопустимы в XML-документах. Вместо того чтобы экранировать все эти символы и получить в результате весьма странный фрагмент XSP, можно воспользоваться тегом CDATA и писать стандартный код Java. Не имея такой возможности, мы создали бы следующий код:

```

<xsp:logic>
    boolean authorOrEditor = false;

    String[] roleValues =
        request.getParameterValues("&quot;userRole&quot;");
    String[] passwordValues =
        request.getParameterValues("&quot;password&quot;");
    if ((roleValues != null) &amp;&amp;
        (passwordValues != null)) {
        String userRole = roleValues[0];
        String password = passwordValues[0];
        if (userRole.equals("author") &amp;&amp;
            password.equals("brett")) {
            authorOrEditor = true;
        } else
        if (userRole.equals("editor") &amp;&amp;
            password.equals("mike")) {
            authorOrEditor = true;
        }
    }
}
...
</xsp:logic>

```

Теперь можно проверить стартовую страницу и получаемый PDF-документ, сгенерированный из данных XML. Если указать в браузере адрес <http://<hostname>:<port>/cocoon/entry.html>, то должен получиться результат, сходный с тем, что показан на рис. 10.8.

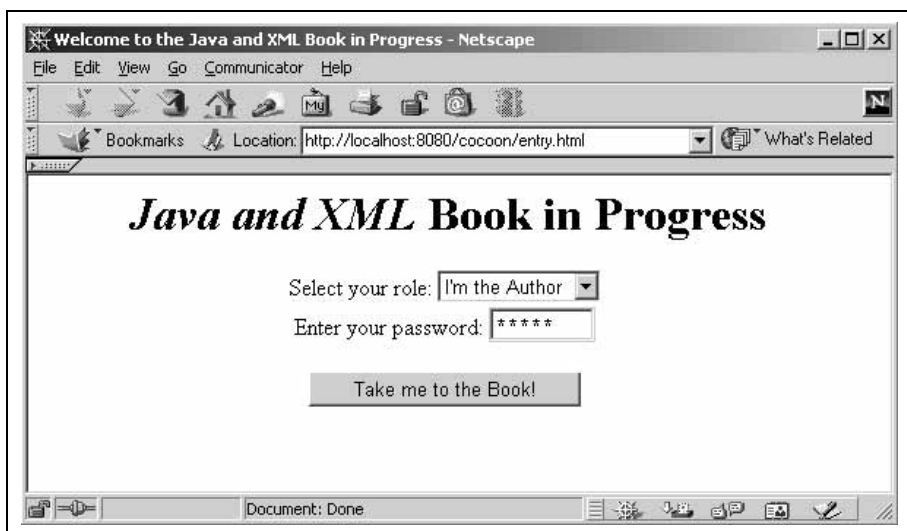


Рис. 10.8. Стартовая страница для XSP-страницы *chapterTen.xml*

Выберите роль автора и укажите пароль «brett» либо выберите роль редактора с паролем «mike». В обоих случаях будет получен PDF-документ, показанный на рис. 10.9.

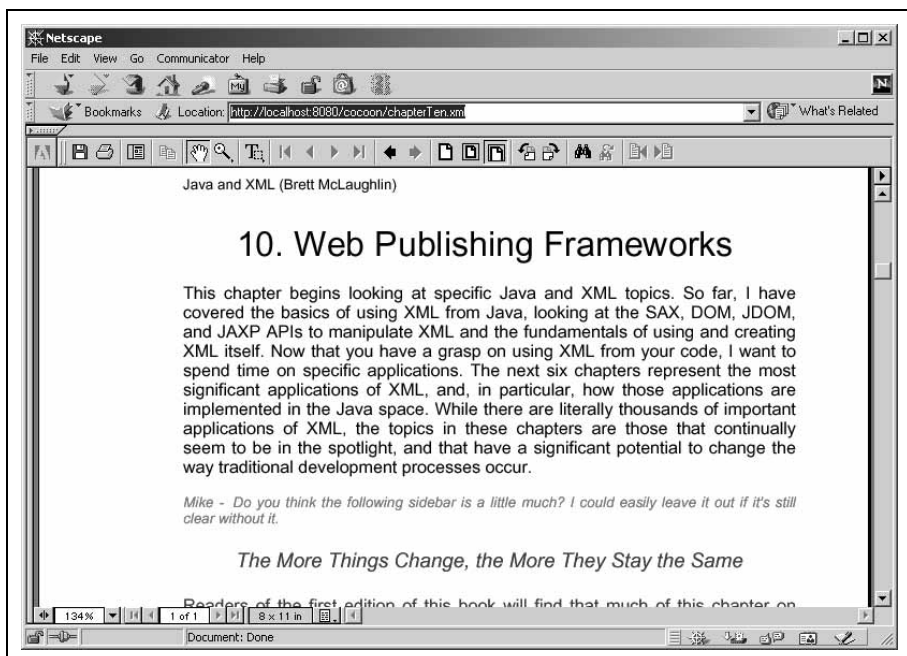


Рис. 10.9. Документ, отображаемый для автора или редактора

Здесь осталось сделать лишь одно – отделить логику страницы от ее информационного наполнения. Как страницы JSP допускают использование JavaBeans для изоляции содержимого и представления от логики компонента приложения, так страницы XSP, в свою очередь, допускают применение *библиотек тегов (tag libraries)*. Тег библиотеки может быть связан с фрагментом логики приложения, выполнение которого происходит, если тег встречается в документе.

Использование библиотек тегов XSP

Помимо отображения комментариев в зависимости от пользователя, XSP-страница должна указывать, что глава находится в черновом состоянии. Можно вывести текущую дату, чтобы показать дату черновика (когда глава завершена, дата фиксируется). Вместо применения внедренного кода на Java для загрузки текущей даты можно просто создать для этой цели пользовательскую библиотеку тегов. Также заслуживает внимания процесс создания элемента XSP, который получает номер и название главы, а затем формирует полный заголовок. Эта функция также будет управлять отображением даты для черновика. Прежде всего, необходимо создать библиотеку тегов, доступную из страницы XSP. Библиотека тегов во многом основана на таблице стилей XSL. Можно начать с каркаса, показанного в примере 10.11, который в качестве результата выдает все, что получает. Сохраните этот каркас в файле *JavaXML.xsp.xml* в подкаталоге *XSL/*. Не забудьте включить объявление пространства имен `javaxml2`, поскольку оно будет использовано для отбора элементов, применяемых в страницах XSP и принадлежащих данному пространству имен.

Пример 10.11. Каркас библиотеки тегов XSP

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xsp="http://www.apache.org/1999/XSP/Core"
  xmlns:javaxml2="http://www.oreilly.com/javaxml2"
  >
  <xsl:template match="xsp:page">
    <xsp:page>
      <xsl:copy>
        <xsl:apply-templates select="@*/"/>
      </xsl:copy>
      <xsl:apply-templates/>
    </xsp:page>
  </xsl:template>
  <xsl:template match="@*|*|text()|processing-instruction()">
    <xsl:copy>
      <xsl:apply-templates
```

```

        select="@*|*|text()|processing-instruction()"/>
    </xsl:copy>
</xsl:template>

</xsl:stylesheet>

```

Создав шаблон для элемента `xsp:page`, можно гарантировать, что в этой таблице стилей – или, в терминах XSP, *таблице логики (logicsheet)* – осуществляется выборка и обработка всех таких элементов. Теперь можно добавить методы Java, вызываемые для шаблонов из этой таблицы логики:

```

<xsl:template match="xsp:page">
  <xsp:page>
    <xsl:copy>
      <xsl:apply-templates select="@*" />
    </xsl:copy>

    <xsp:structure>
      <xsp:include>java.util.Date</xsp:include>
      <xsp:include>java.text.SimpleDateFormat</xsp:include>
    </xsp:structure>

    <xsp:logic>
      private static String getDraftDate() {
        return (new SimpleDateFormat("MM/dd/yyyy"))
          .format(new Date());
      }

      private static String getTitle(int chapterNum,
        String chapterTitle) {
        return chapterNum + ". " + chapterTitle;
      }
    </xsp:logic>

    <xsl:apply-templates/>
  </xsp:page>
</xsl:template>

```

Здесь представлено несколько новых элементов XSP. Во-первых, структура `xsp:structure`, которая содержит несколько инструкций `xsp:include`. Они работают точно так же, как их аналог в Java – оператор `import`, обеспечивая доступ к указанным классам Java по краткому имени (а не по полному, содержащему название пакета). Затем таблица логики определяет и реализует два метода: один из них формирует заголовок на основе номера главы и ее названия, а другой возвращает текущую дату в виде отформатированной строки (`String`). Эти методы доступны для всех элементов из данной таблицы логики.

Теперь создадим элемент, который определяет, когда результат выполнения XSP должен заменять элемент XML. Пространство имен, связанное с префиксом `javaxml2`, уже определено в корневом элементе документа, и его можно использовать в качестве пространства имен

для элементов библиотеки тегов. Добавьте следующий шаблон в таблицу логики:

```

<!-- Создание отформатированного заголовка -->
<xsl:template match="javaxml2:draftTitle">
  <xsp:expr>getTitle(<xsl:value-of select="@chapterNum" />,
    "<xsl:value-of select="@chapterTitle" />")
  </xsp:expr> (<xsp:expr>getDraftDate()</xsp:expr>)
</xsl:template>

<xsl:template match="@*|*|text()|processing-instruction()">
  <xsl:copy>
    <xsl:apply-templates
      select="@*|*|text()|processing-instruction()"/>
  </xsl:copy>
</xsl:template>

```

Когда в документе, из которого доступна эта библиотека тегов, встречается элемент `javaxml2:draftTitle` (или просто `draftTitle`, если пространство имен по умолчанию связано с <http://www.oreilly.com/javaxml2>), то значение, возвращаемое методом `getTitle()`, предваряет значение, возвращаемое методом `getDraftDate()`. Элемент `javaxml2:draftTitle` требует также наличия двух атрибутов: номера главы и ее названия. Помещая вызов метода внутрь пары тегов `<xsp:expr>`, мы указываем процессору XSP, что происходит вызов определенного метода. Чтобы указать, что второй аргумент (название главы) является строкой, мы заключаем его в кавычки. Номер главы должен интерпретироваться как целочисленное значение, так что он остается без кавычек.

Закончив составление таблицы логики XSP (которая также доступна на сайте этой книги), необходимо сделать ее доступной для системы Coooon. Это можно сделать одним из двух способов. Первый – указать местоположение файла в виде URI, который позволяет среде исполнения сервлетов (а следовательно, системе Coooon) найти таблицу логики. Например, чтобы добавить таблицу логики XSP к набору ресурсов системы Coooon с помощью URI соответствующего файла в Unix-системе, можно добавить следующую строку в файл `cocoon.properties`:

```

# Установка библиотек, связанных с данным пространством имен.
# Используйте синтаксис:
# processor.xsp.logicsheet.<namespace-tag>.<language> = URL для файла
# где "URL для файла" обычно начинается с file://, если пользовательская
# библиотека находится в вашей файловой системе
processor.xsp.logicsheet.context.java = resource://org/apache/cocoon/
processor.xsp/library/java/context.xsl
processor.xsp.logicsheet.cookie.java = resource://org/apache/cocoon/
processor.xsp/library/java/cookie.xsl
processor.xsp.logicsheet.global.java = resource://org/apache/cocoon/
processor.xsp/library/java/global.xsl

```

```

processor.xsp.logicsheet.request.java = resource://org/apache/cocoon/
processor/xsp/library/java/request.xml
processor.xsp.logicsheet.response.java = resource://org/apache/cocoon/
processor/xsp/library/java/response.xml
processor.xsp.logicsheet.session.java = resource://org/apache/cocoon/
processor/xsp/library/java/session.xml
processor.xsp.logicsheet.util.java     =
    resource://org/apache/cocoon/processor/xsp/library/java/util.xml
processor.xsp.logicsheet.sql.java      =
    resource://org/apache/cocoon/processor/xsp/library/sql/sql.xml
processor.xsp.logicsheet.esql.java     =
    resource://org/apache/cocoon/processor/xsp/library/sql/esql.xml
processor.xsp.logicsheet.fp.java       =
    resource://org/apache/cocoon/processor/xsp/library/fp/fp.xml

processor.xsp.library.JavaXML.java =
    file:///usr/local/jakarta-tomcat/webapps/cocoon/XSL/JavaXML.xsp.xml

```

Для систем Windows это может выглядеть следующим образом:

```

# Установка библиотек, связанных с данным пространством имен.
# Используйте синтаксис:
#   processor.xsp.logicsheet.<namespace-tag>.<language> = URL для файла
# где "URL для файла" обычно начинается с file://, если пользовательская
# библиотека находится в вашей файловой системе
processor.xsp.logicsheet.context.java = resource://org/apache/cocoon/
processor/xsp/library/java/context.xml
processor.xsp.logicsheet.cookie.java  = resource://org/apache/cocoon/
processor/xsp/library/java/cookie.xml
processor.xsp.logicsheet.global.java  = resource://org/apache/cocoon/
processor/xsp/library/java/global.xml
processor.xsp.logicsheet.request.java = resource://org/apache/cocoon/
processor/xsp/library/java/request.xml
processor.xsp.logicsheet.response.java = resource://org/apache/cocoon/
processor/xsp/library/java/response.xml
processor.xsp.logicsheet.session.java = resource://org/apache/cocoon/
processor/xsp/library/java/session.xml
processor.xsp.logicsheet.util.java    =
    resource://org/apache/cocoon/processor/xsp/library/java/util.xml
processor.xsp.logicsheet.sql.java     =
    resource://org/apache/cocoon/processor/xsp/library/sql/sql.xml
processor.xsp.logicsheet.esql.java    =
    resource://org/apache/cocoon/processor/xsp/library/sql/esql.xml
processor.xsp.logicsheet.fp.java      =
    resource://org/apache/cocoon/processor/xsp/library/fp/fp.xml

processor.xsp.library.javaxml2.java =
    file:///C:/java/jakarta-tomcat/webapps/cocoon/XSL/JavaXML.xsp.xml

```

Хотя такое решение вполне подходит для тестирования, оно не слишком эффективно в смысле уменьшения зацепления таблиц логики

и среды исполнения сервлетов. Кроме того, оно влечет за собой значительные накладные расходы, связанные с сопровождением: чтобы сделать доступной новую таблицу логики, придется добавить новые строки в файл свойств *cocoon.properties*. Альтернативный метод загрузки таблиц логики связан с описанием ресурса в *classpath*-переменной среды исполнения сервлетов. Мы получаем возможность поместить все пользовательские таблицы логики в *jar*-файл и сослаться на этот файл в пути к классам (в Tomcat это делается простым добавлением архива в каталог *lib/!*) среды исполнения сервлетов. Кроме того, в этот *jar*-файл можно помещать новые таблицы логики, а значит, есть возможность централизованного хранения таблиц логики XSP. Для создания *jar*-файла, содержащего нашу таблицу логики, выполните следующую команду в подкаталоге *XSL/* корневого каталога документов своего веб-сервера:

```
jar cvf logicsheets.jar JavaXML.xsp.xml
```

Перенесите созданный архив *logicsheets.jar* в каталог *<TOMCAT_HOME>/lib/* (в нем уже хранятся остальные библиотеки Cocoon). Это гарантирует, что Tomcat загрузит библиотеку при запуске. Обеспечив доступ к таблице логики, можно теперь указать системе Cocoon, где следует искать ссылки на пространство имен *javaxml2* в страницах XSP. Отредактируйте файл *cocoon.properties*; найдите раздел, в котором перечислены различные XSP-ресурсы системы Cocoon, и добавьте новую ссылку на таблицу стилей:

```
# Установка библиотек, связанных с заданным пространством имен.
# Используйте синтаксис:
#   processor.xsp.logicsheet.<namespace-tag>.<language> = URL для файла
# где "URL для файла" обычно начинается с file://, если пользовательская
# библиотека находится в вашей файловой системе
processor.xsp.logicsheet.context.java = resource://org/apache/cocoon/
processor/xsp/library/java/context.xml
processor.xsp.logicsheet.cookie.java = resource://org/apache/cocoon/
processor/xsp/library/java/cookie.xml
processor.xsp.logicsheet.global.java = resource://org/apache/cocoon/
processor/xsp/library/java/global.xml
processor.xsp.logicsheet.request.java = resource://org/apache/cocoon/
processor/xsp/library/java/request.xml
processor.xsp.logicsheet.response.java = resource://org/apache/cocoon/
processor/xsp/library/java/response.xml
processor.xsp.logicsheet.session.java = resource://org/apache/cocoon/
processor/xsp/library/java/session.xml
processor.xsp.logicsheet.util.java =
    resource://org/apache/cocoon/processor/xsp/library/java/util.xml
processor.xsp.logicsheet.sql.java =
    resource://org/apache/cocoon/processor/xsp/library/sql/sql.xml
processor.xsp.logicsheet.esql.java =
    resource://org/apache/cocoon/processor/xsp/library/sql/esql.xml
processor.xsp.logicsheet.fp.java =
```

```
resource://org/apache/cocoon/processor/xsp/library/fp/fp.xml
processor.xsp.logicsheet.javaxml2.java = resource://JavaXML.xsp.xml
```

Данная таблица логики не располагается ни в одном из подкаталогов архива *logicsheets.jar*, поэтому можно просто использовать ее имя в качестве идентификатора ресурса. Наконец, перезапустите среду исполнения сервлетов (это также гарантирует, что Tomcat автоматически загрузит новую библиотеку). При этом файл *cocoon.properties* будет загружен заново, и таблица логики станет доступной для использования. Поскольку для обработки запросов применяется система Coooon, любая XSP-страница, в которой объявлено, что она использует пространство имен *javaxml2*, получит доступ к таблице логики, описанной как библиотека *javaxml2*. Таким образом, в XSP-страницу следует добавить объявление пространства имен *javaxml2*:

```
<?xml version="1.0"?>
<?cocoon-process type="xsp"?>
<?cocoon-process type="xslt"?>
<?xml-stylesheet href="XSL/JavaXML.fo.xml" type="text/xsl"?>
<xsp:page language="java"
  xmlns:xsp="http://www.apache.org/1999/XSP/Core"
  xmlns:javaxml2="http://www.oreilly.com/javaxml2"
>
<book>
  <!-- Содержимое книги -->
</book>
</xsp:page>
```

Теперь, когда библиотека тегов доступна для использования, можно, наконец, добавить элемент *javaxml2:draftTitle* в XML-документ *chapterTen.xml*:

```
<contents>
  <chapter title="Web Publishing Frameworks" number="10">
    <javaxml2:draftTitle chapterNum="10"
      chapterTitle="Web Publishing Framework" />
  ...
```

Замените жестко закодированный заголовок главы элементом, определенным в библиотеке тегов XSP, изменив следующим образом таблицу стилей *JavaXML.fo.xml*:

```
<xsl:template match="chapter">
  <fo:block font-size="24pt"
    text-align-last="center"
    space-before.optimum="24pt">
<!--
  <xsl:value-of select="@number" />.
  <xsl:value-of select="@title" />
-->
```

```
<xsl:apply-templates/>
</fo:block>
</xsl:template>
```

В результате должен быть сгенерирован заголовок с номером главы, ее названием и датой создания черновика. При обращении к новой версии XSP-страницы выдается результат, показанный на рис. 10.10.

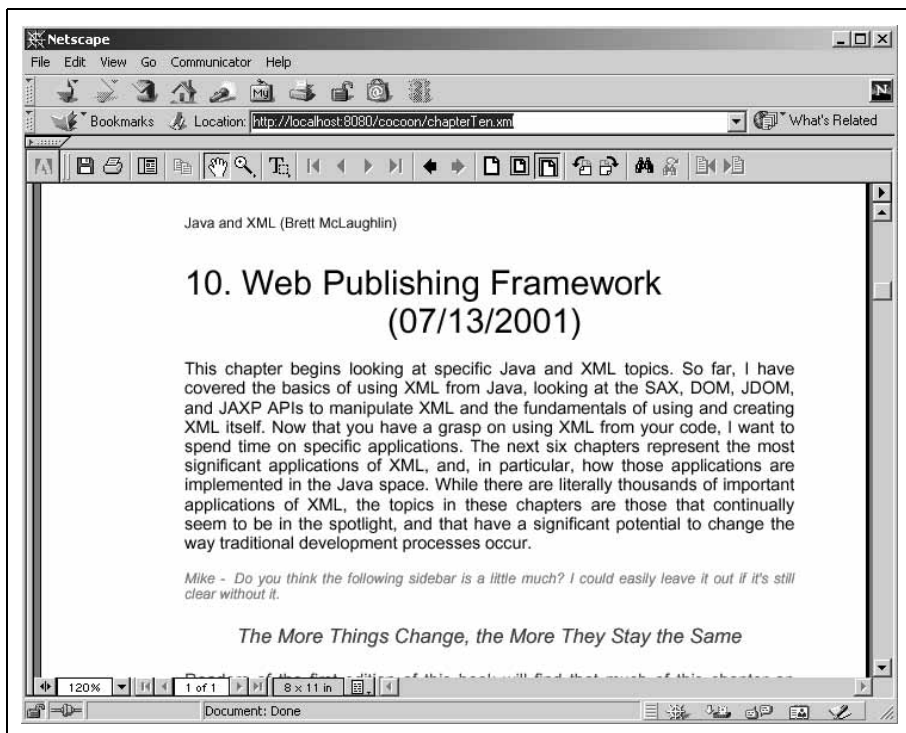


Рис. 10.10. Вывод, получаемый при использовании библиотеки тегов XSP

Мы лишь коснулись азов технологии XSP. Но даже этот простой пример позволяет преобразовать заголовок в другой формат, когда глава будет завершена, не меняя ни содержания, ни представления страницы, а внося изменение лишь в таблицу логики XSP. Аналогичным образом XSP позволяет создавать очень строгие границы, разделяющие представление, информационное наполнение и логику приложения. Серверные компоненты Java, такие как Enterprise JavaBeans, могут добавить в это «уравнение» еще и бизнес-логику. Лучше избегать менее гибких решений, подобных JSP, которые довольно жестко связаны с HTML и представлением данных. Применение XSP позволяет сократить зацепление компонентов и, следовательно, представляет собой более предпочтительное решение в смысле разработки приложений. XSP также обещает стать ключевой технологией в системе Cocoon 2.0, которую мы сейчас рассмотрим.

Сосоон 2.0 и выше

Следующее поколение Сосоон, система Сосоон 2.0, должна стать гигантским шагом вперед в области систем веб-публикации. Система Сосоон 1.x, основанная главным образом на связке XML+XSL, по-прежнему имеет ряд серьезных ограничений. Прежде всего, она не столь значительно сокращает расходы на сопровождение крупных сайтов. И хотя один XML-документ может быть преобразован в различные клиентские форматы, количество документов от этого не сокращается. В общем случае результатом являются либо длинные URI (скажем, `/content/publishing/books/javaxml/contents.xml`), либо большое количество виртуальных отображений путей (`/javaxml` соответствующий `/content/publishing/books/javaxml`), либо комбинация обоих случаев. Кроме того, достаточно сложно достичь – и еще сложнее поддерживать – строгое разделение информационного наполнения, представления и логики.

Сосоон 2 концентрирует внимание на совершенствовании разграничения между этими различными уровнями и, следовательно, сокращении расходов на сопровождение. Основой архитектуры Сосоон 2.0 является технология XSP. Кроме того, карта сайта позволяет скрыть различия между XSP, XML и статическими страницами HTML от любопытствующего пользователя. В системе Сосоон 2 появится более совершенный вариант предварительной компиляции и более эффективное управление памятью, что даст ей еще больше преимуществ перед системой Сосоон 1.x, чем Сосоон 1.x имела по сравнению со стандартным веб-сервером.

Среда исполнения сервлетов: отображения

Одно из значительных изменений в Сосоон 2 состоит в том, что больше нет необходимости использовать примитивный механизм отображений для XML-документов. Хотя эта методика очень хорошо работает в модели 1.x, она полностью оставляет управление документами, отличными от XML, на совести веб-мастера, которым может оказаться совсем не тот человек, который отвечает за XML-документы. Система Сосоон 2 стремится принять на себя управление целым веб-сайтом, поэтому основной сервлет Сосоон (которым в версии 2.0 является `org.apache.cocoon.servlet.CocoonServlet`) обычно ставится в соответствие URI вроде `/Cocoon`. В целях полного контроля за сайтом основной сервлет может быть связан с корневым каталогом веб-сервера (т. е. «/»). После этого URL запрашиваемых документов должны учитывать отображение: например `http://myHost.com/Cocoon/myPage.xml` или `http://myHost.com/Cocoon/myDynamicPage.xsp`.

При наличии такого отображения можно совместно хранить документы XML и статические документы HTML. Это делает возможным управление всеми файлами сервера централизованно одним человеком

или группой. Если HTML-, WML- и XML-документы необходимо хранить в одном и том же каталоге, никакой путаницы не происходит, и можно использовать единообразные URI. Cocoon 2 будет с таким же успехом выдавать HTML-документ, как и документ любого другого типа. В результате привязки корневого каталога сервера к системе публикации Cocoon система веб-публикации становится фактически невидимой для клиента.

Карта сайта

Еще одно важное нововведение в Cocoon 2 – это *карта сайта (sitemap)*. Карта сайта обеспечивает в системе Cocoon централизованное администрирование веб-сайта. Cocoon использует эту карту сайта для принятия решений о том, как обрабатывать получаемые запросы к URI. Например, когда Cocoon получает запрос вида *http://myCocoon-Site.com/Cocoon/javaxml/chapterOne.html*, сервлет системы Cocoon анализирует запрос и определяет, что реальным запрашиваемым URI является */javaxml/chapterOne.html*. Предположим, однако, что файл *chapterOne.html* должен соответствовать не статическому HTML-файлу, а преобразованию XML-документа (как в наших предыдущих примерах). Карта сайта может достаточно просто учесть такую ситуацию! Взгляните на пример 10.12.

Пример 10.12. Простая карта сайта Cocoon 2

```
<sitemap>
  <process match="/javaxml/*.html">
    <generator type="file" src="/docs/javaxml/*.xml"
    <filter type="xslt">
      <parameter name="stylesheet" value="/styles/JavaXML.html.xsl"/>
    </filter>
    <serializer type="html"/>
  </process>

  <process match="/javaxml/*.pdf">
    <generator type="file" src="/docs/javaxml/*.xml"
    <filter type="xslt">
      <parameter name="stylesheet" value="/styles/JavaXML.pdf.xsl"/>
    </filter>
    <serializer type="fop"/>
  </process>
</sitemap>
```

В этом примере система Cocoon сопоставляет URI */javaxml/chapterOne.html* с инструкцией карты сайта */javaxml/*.html*. Она определяет, что указанный URI соответствует реальному файлу и что файл, содержащий исходные данные, следует определить, используя отображение */docs/javaxml/*.xml*, что дает */docs/javaxml/chapterOne.xml* (имя файла, который мы хотим преобразовать). Затем применяется фильтр XSLT – применяемая таблица стилей *JavaXML.html.xsl* также задана

в карте сайта. Далее преобразованные данные выдаются пользователю. Кроме того, XML-файл может быть файлом XSP, который обрабатывается до того, как преобразуется в XML, и затем форматируется.

Таким же образом можно вывести данные в формате PDF по запросу <http://myCocoonSite.com/Cocoon/javaxml/chapterOne.pdf>, добавив всего несколько строк в приведенную выше карту сайта. Это означает, что можно полностью удалить инструкции обработки из индивидуальных XML-документов, что представляет собой значительное изменение по сравнению с Сосооп 1.x. В первую очередь, в зависимости от каталога, в котором размещены файлы, можно осуществлять единообразное применение таблиц стилей и единообразную обработку. Одно лишь создание XML-документа и помещение его в каталог `/docs/javaxml/` в данном примере означает, что доступ к документу возможен как в формате HTML, так и в PDF. Так же легко можно будет изменить таблицу стилей, применяемую для набора документов, что представляет собой трудную и утомительную задачу в Сосооп 1.x. Вместо того чтобы вносить изменения в каждый XML-документ, необходимо изменить всего лишь одну строку в карте сайта.

Карта сайта Сосооп еще находится в стадии разработки, и ко времени выхода окончательной версии Сосооп 2.0 в ее формат и структуру, вероятно, будут внесены некоторые дополнительные изменения и усовершенствования. Чтобы быть в курсе последних событий, подпишитесь на списки рассылки cocoon-users@xml.apache.org и cocoon-dev@xml.apache.org. Подробности об участии в этих списках и в проекте Сосооп можно найти на сервере проекта Apache XML <http://xml.apache.org>.

Генераторы и процессоры

И последнее усовершенствование, которое должен включать Сосооп 2, – это прекомпилированные и событийно-ориентированные *генераторы* (*producers*) и *процессоры* (*processors*). В системе Сосооп генератор управляет преобразованием URI запроса в поток, связанный с XML-документом. Далее процессор получает входной поток (в настоящее время XML-документ, представленный в виде дерева DOM) и преобразует его в результат, пригодный для чтения клиентом. Мы не рассматриваем генераторы и процессоры для модели Сосооп 1.x, поскольку в Сосооп 2.0 они будут серьезно переработаны – любые генераторы и процессоры, используемые в настоящее время, вероятнее всего станут неприменимыми в системе Сосооп 2.0.

В системе Сосооп 2 делается переход от применения для этих структур модели DOM к SAX, которая более тесно связана с событиями, оболочкой для которой, как и прежде, будет модель DOM. Поскольку в версиях 1.x генератор должен был хранить XML-документ в памяти, соответствующая структура DOM могла становиться крайне большой. В конце концов, это приводило к истощению системных ресурсов, осо-

бенно при выполнении таких сложных задач, как крупномасштабные преобразования или работа с объектами форматирования (создание документов в формате PDF). По этим причинам в системе Cocomo 2 модель DOM будет играть роль простой оболочки для событий SAX, сообщая генераторам и процессорам легкость и эффективность.

Кроме того, в виде генераторов и процессоров будут реализованы прекомпилированные версии других форматов. Например, таблицы стилей XSL могут быть прекомпилированы в процессоры, а страницы XSP – в генераторы. Это еще больше увеличивает производительность и в то же время снимает нагрузку с клиента. Эти и другие изменения происходят по правилам компонентной модели, что делает систему Cocomo очень гибкой и легко внедряемой. Чтобы быть в курсе последних изменений, следите за информацией, публикуемой на сайте проекта Cocomo.¹

Что дальше?

В следующей главе рассматривается технология, обеспечивающая использование XML как формата данных в важной модели запроса и ответа – XML-RPC. Вызов удаленных процедур с применением XML позволяет клиентам распределенной системы выполнять задачи на сервере или серверах, расположенных в другом сегменте сети. До последнего времени популярность RPC была низкой, главным образом, из-за волны RMI-технологий в мире Java (в первую очередь, это EJB). Однако, используя XML как формат данных, XML-RPC стал новым решением для многих проблем, которые нельзя решить ясно и эффективно другими методами. В следующей главе мы рассмотрим технологию XML-RPC и, в частности, ее реализацию на Java.

¹ <http://xml.apache.org/cocomo>. – Примеч. науч. ред.